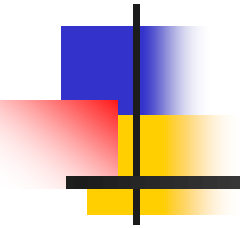
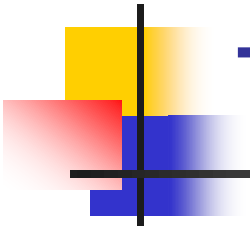


Algorithms and Data Structures
Lecture XV

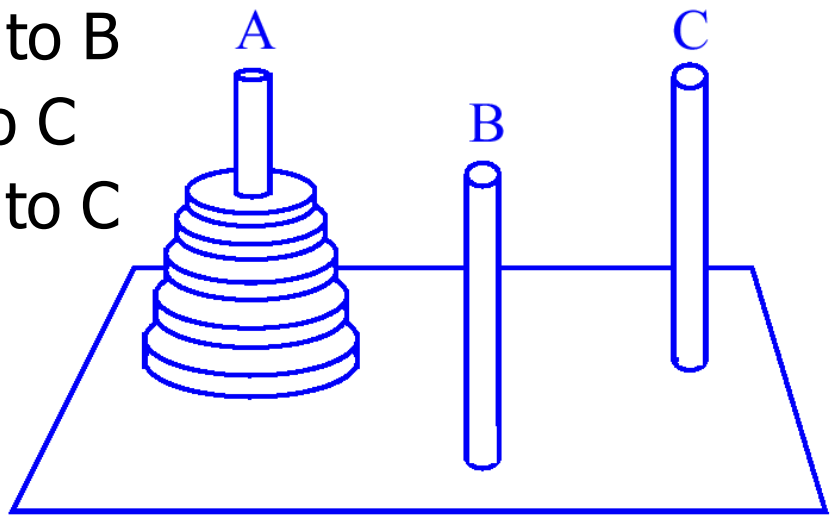


This Lecture

- Tractable and intractable alg. Problems
  - What is a ”reasonable” running time?
  - NP problems, examples
  - NP-complete problems and polynomial reducability
- What have we learned?

Towers of Hanoi

- **Goal:** transfer all *n* disks from peg A to peg C
- Rules:
  - move one disk at a time
  - never place larger disk above smaller one
- Recursive solution:
  - transfer $n - 1$ disks from A to B
  - move largest disk from A to C
  - transfer $n - 1$ disks from B to C
- Total number of moves:
  - $T(n) = 2T(n - 1) + 1$



Towers of Hanoi (2)

- Recurrence relation:

  $T(n) = 2\,T(n-1) + 1$

  $T(1) = 1$

- Solution by unfolding:

  $T(n) = 2\,(2\,T(n-2) + 1) + 1 =$

  $= 4\,T(n-2) + 2 + 1 =$

  $= 4\,(2\,T(n-3) + 1) + 2 + 1 =$

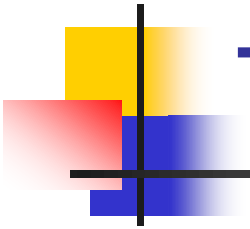
  $= 8\,T(n-3) + 4 + 2 + 1 = \ldots$

  $= 2^i\,T(n-i) + 2^{i-1} + 2^{i-2} + \ldots + 2^1 + 2^0$

- the expansion stops when $i = n - 1$

  $T(n) = 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2^1 + 2^0$

Towers of Hanoi (3)



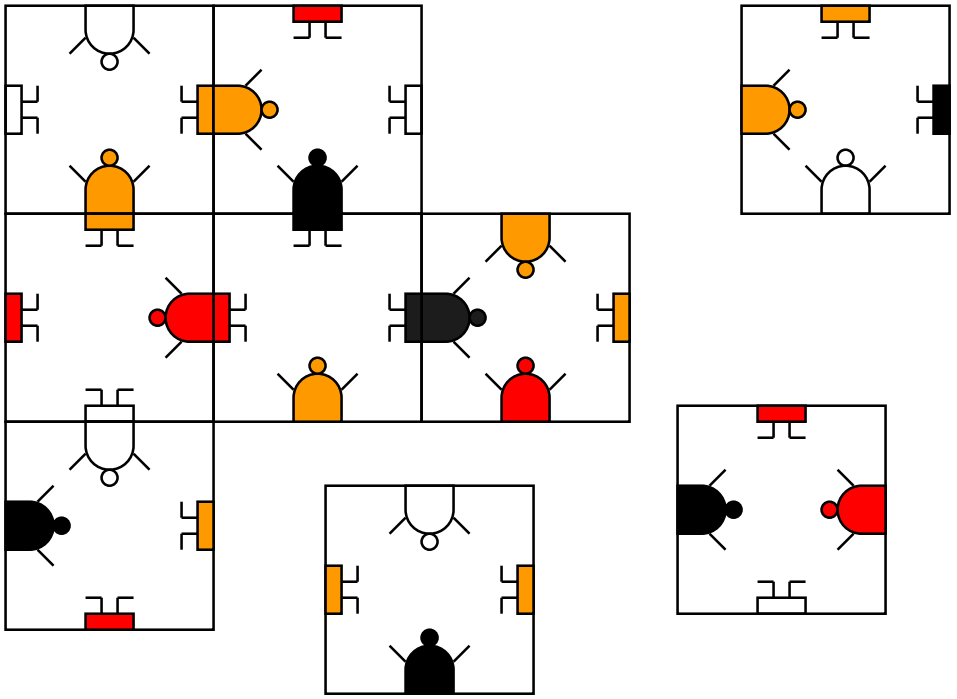
- This is a **geometric sum**, so that we have

  $T(n) = 2^n - 1 = O(2^n)$

- The running time of this algorithm is **exponential** (k^n) rather than **polynomial** (n^k)
- Good or bad news?
  - the Tibetans were confronted with a tower problem of 64 rings...
  - assuming one could move **1 million rings per second**, it would take **half a million years** to complete the process...

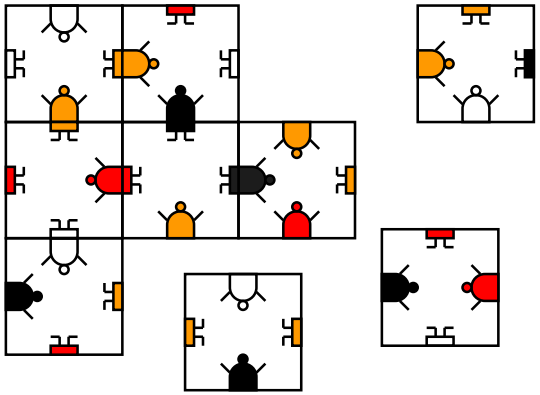
Monkey Puzzle

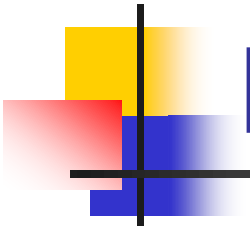
- Are such long running times linked to the size of the solution of an algorithm?
  - No! To show that, we in the following consider only TRUE/FALSE or yes/no problems – **decision problems**
- Nine square cards with imprinted "monkey halfs"
- The goal is to arrange the cards in 3x3 square with matching halfs...



More Monkey...

- Assumption: orientation is fixed
- **Does any MxM arrangement exist that fulfills the matching criterion?**
- Brute-force algorithm would take ***n*! times** to verify whether a solution exists
  - assuming $n = 25$, it would take 490 billion years on a one-million-per-second arrangements computer to verify whether a solution exists

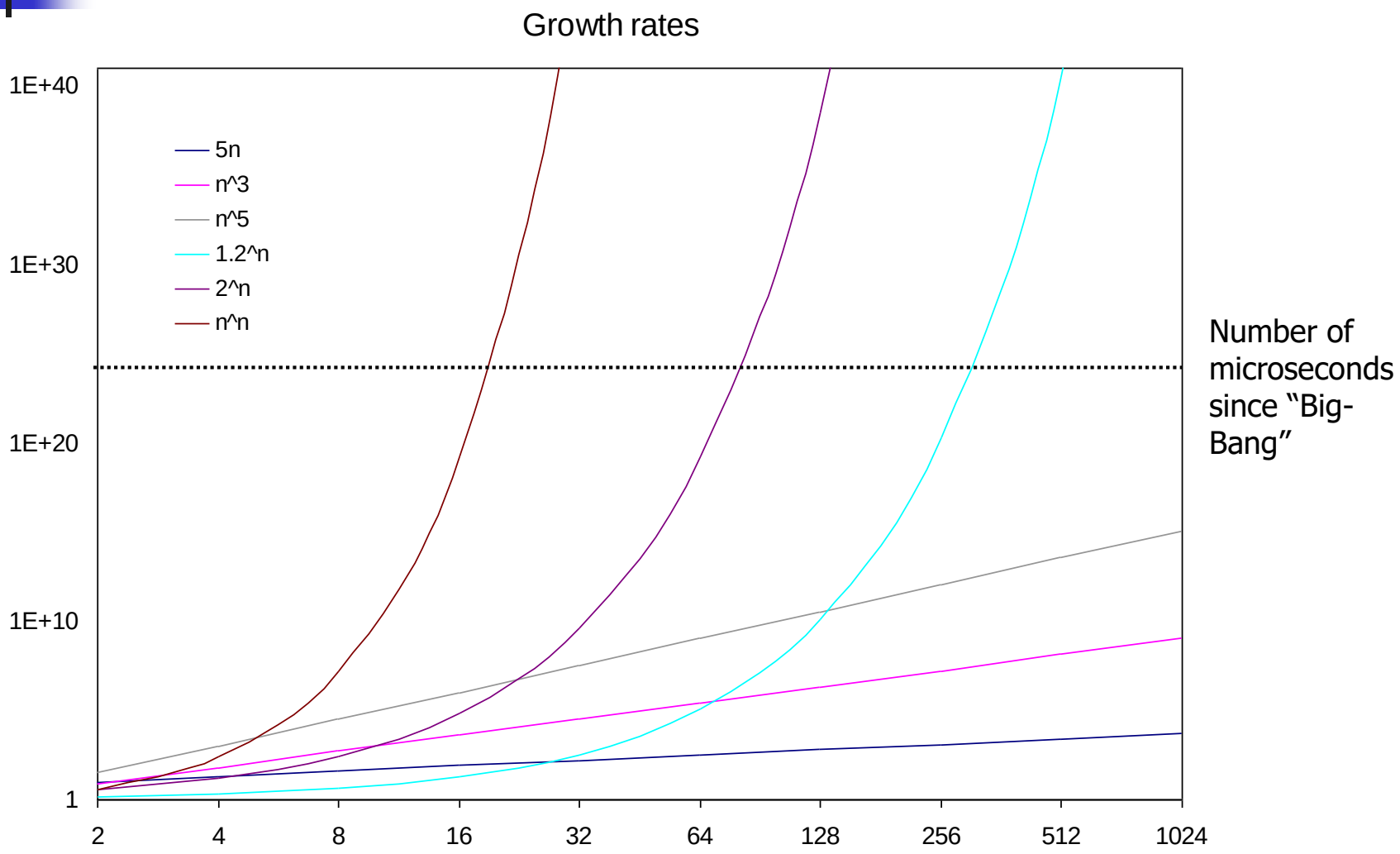




Monkey (3)

- Improving the algorithm
  - discarding partial arrangements
  - etc.
- A smart algorithm would still take a couple of thousand years in the worst case
- Is there an easier way to find solutions? MAYBE! But nobody has found them, yet! (room for smart students...)

Reasonable vs. Unreasonable

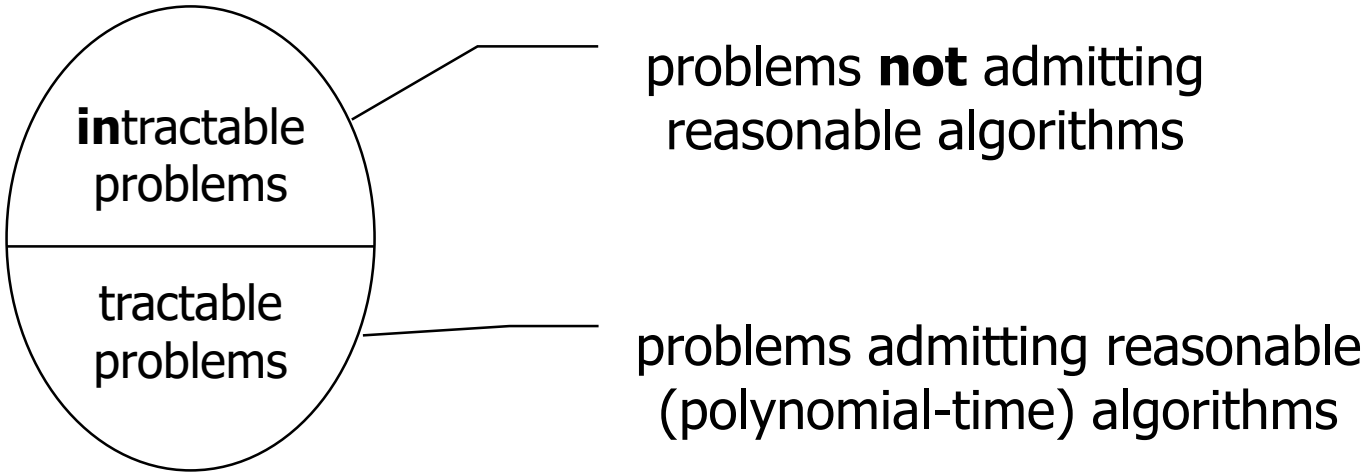


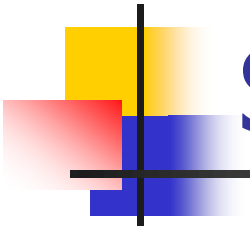
Reasonable vs. Unreasonable

	function/ n	10	20	50	100	300
Polynomial	n^2	1/10,000 second	1/2,500 second	1/400 second	1/100 second	9/100 second
Polynomial	n^5	1/10 second	3.2 seconds	5.2 minutes	2.8 hours	28.1 days
Exponential	2^n	1/1000 second	1 second	35.7 years	400 trillion centuries	a 75 digit-number of centuries
Exponential	n^n	2.8 hours	3.3 trillion years	a 70 digit-number of centuries	a 185 digit-number of centuries	a 728 digit-number of centuries

Reasonable vs. Unreasonable

- ”Good”, reasonable algorithms
  - algorithms bound by a polynomial function n^k
  - **Tractable problems**
- ”Bad”, unreasonable algorithms
  - algorithms whose running time is above n^k
  - **Intractable problems**



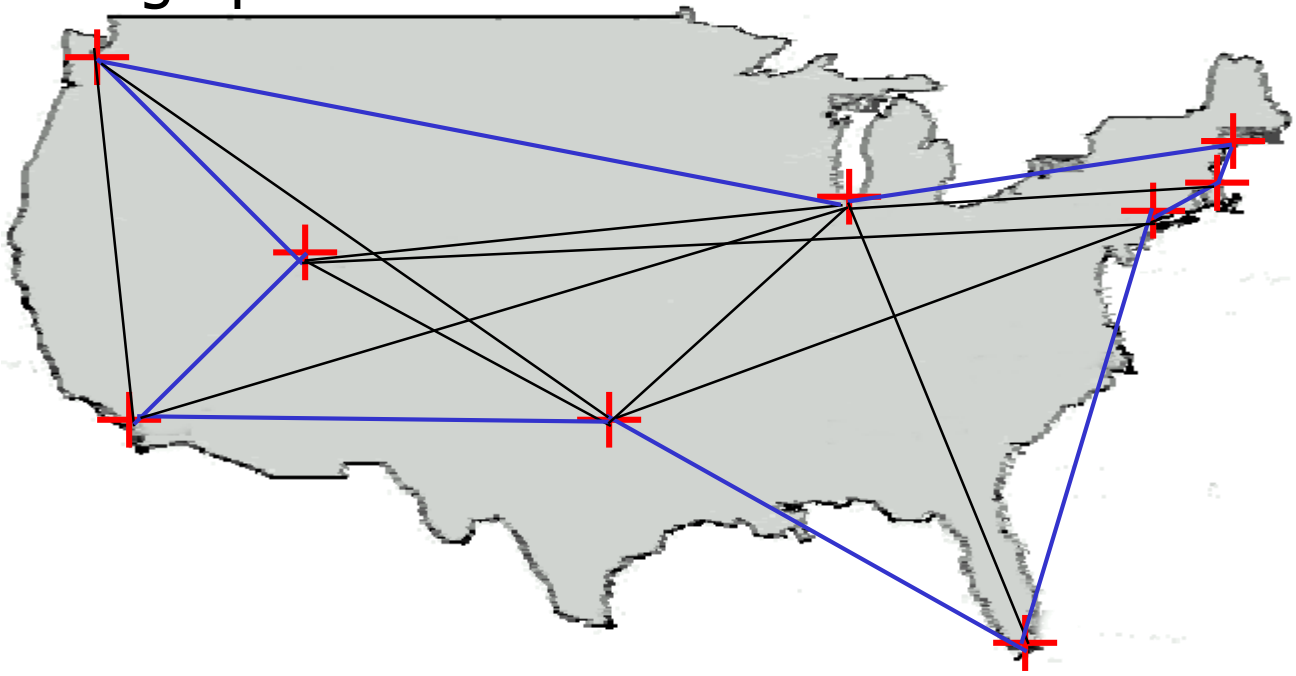


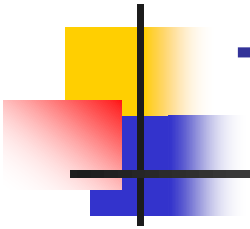
So What!

- Computers become faster every day
  - insignificant (a constant) compared to exp. running time
- Maybe the Monkey puzzle is just one specific one, we could simply ignore
  - the monkey puzzle falls into a category of problems called NPC (NP complete) problems (~1000 problems)
  - all admit **unreasonable** solutions
  - **not known** to admit **reasonable** ones…

Traveling Salesman Problem

- A traveling salesperson needs to visit *n* cities
- Is there a route of at most *d* length? (decision problem)
  - Optimization-version asks to find a shortest path in a weighted graph



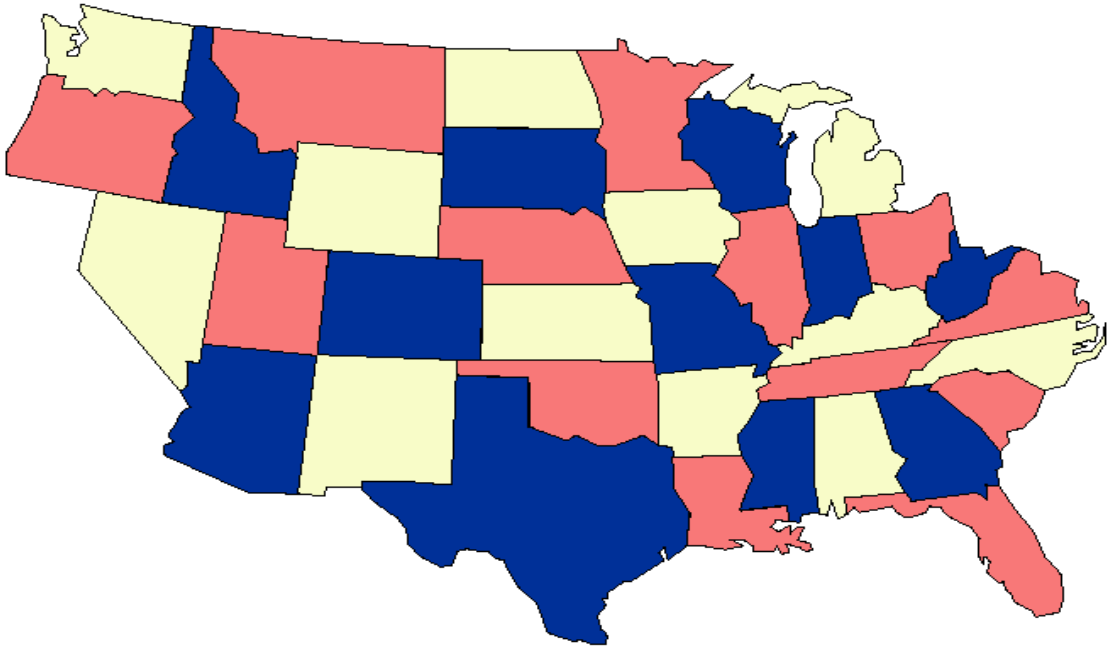


TSP Algorithms

- Naive solutions take $n!$ time in worst-case, where n is the number of edges of the graph
- No polynomial-time algorithms are known
  - TSP is an NP-complete problem
- Longest Path problem between A and B in a weighted grapah is also NP-complete
  - Remember the running time for the shortest path problem

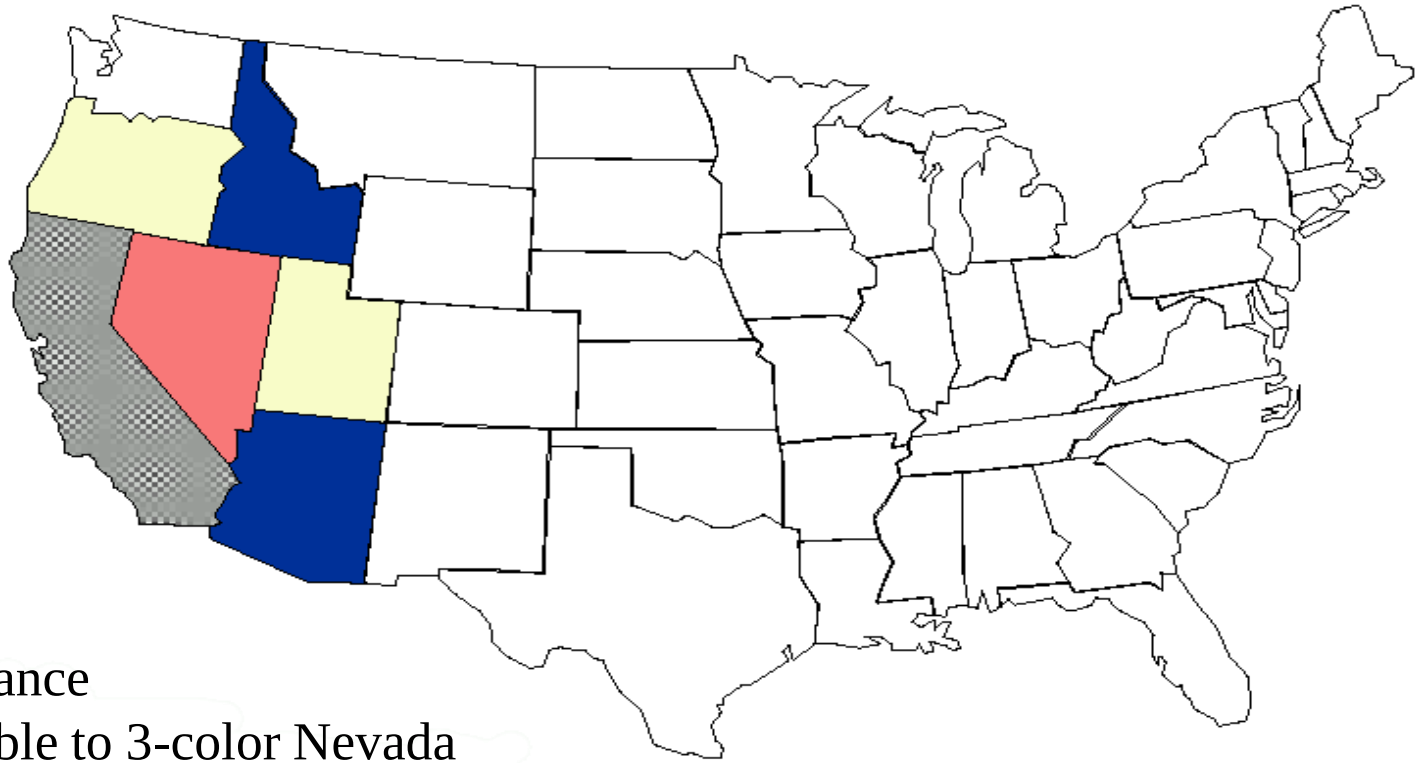
Coloring Problem (COLOR)

- 3-color
  - given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color



YES instance

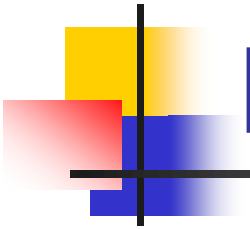
Coloring Problem (2)



NO instance
Impossible to 3-color Nevada and bordering states!

Coloring Problem (3)

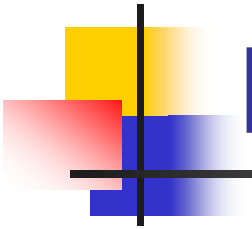
- Any map can be **4-colored**
- Maps that contain no points that are the junctions of an odd number of states can be **2-colored**
- No polynomial algorithms are known to determine whether a map can be **3-colored** – it's an NP-complete problem



Determining Truth (SAT)

- Determine the truth or falsity of logical sentences in a simple logical formalism called **propositional calculus**
- Using the logical connectives (&-and, $\vee$-or, ~-not, $\rightarrow$-implies) we compose expressions such as the following

  $\sim(E \rightarrow F) \;\&\; (F \vee (D \rightarrow \sim E))$
- The algorithmic problem calls for determining the **satisfiability** of such sentences
  - e.g., E = true, D and F = false

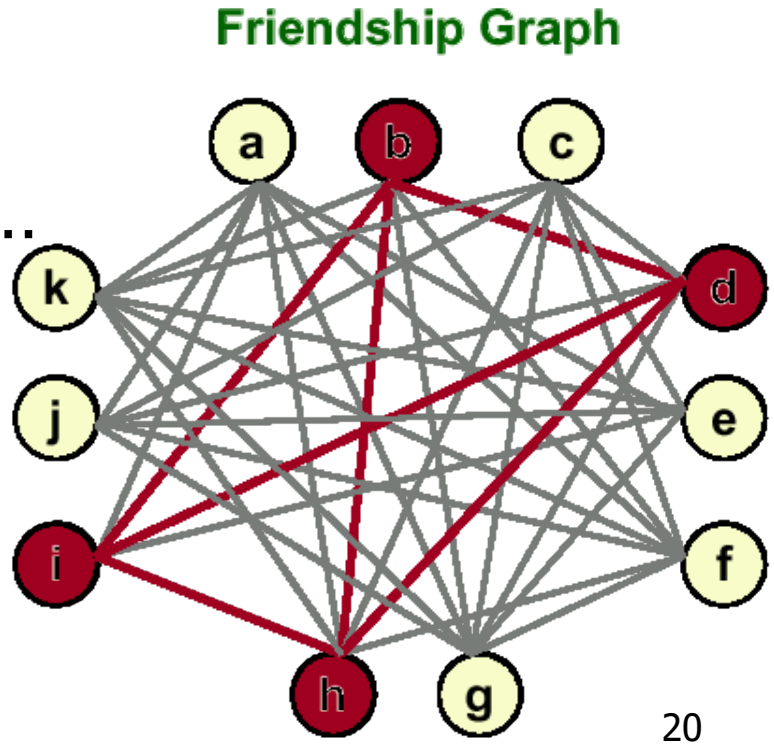


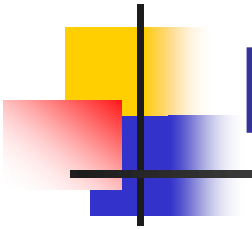
Determining Truth (SAT)

- Exponential time algorithm on n = the number of distinct elementary assertions ($O(2^n)$)
- Best known solution, problem is in NP-complete class!

CLIQUE

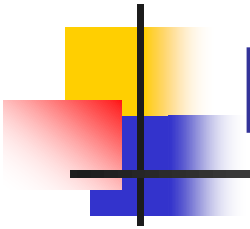
- Given n people and their pairwise relationships, is there a group of s people such that every pair in the group knows each other
  - people: a, b, c, …, k
  - friendships: (a,e), (a,f),…
  - clique size: s = 4?
  - YES, {b, d, i, h}is a certificate!





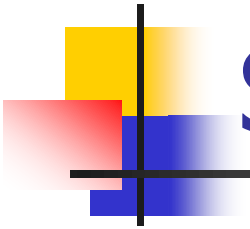
P

- Definition of P:
  - Set of all decision problems solvable in polynomial time on a deterministic Turing machine
- Examples:
  - MULTIPLE: Is the integer y a multiple of x?
    - YES: (x, y) = (17, 51).
  - RELPRIME: Are the integers x and y relatively prime?
    - YES: (x, y) = (34, 39).
  - MEDIAN: Given integers x_1 , …, x_n , is the median value < M?
    - YES: (M, x_1 , x_2 , x_3 , x_4 , x_5) = (17, 2, 5, 17, 22, 104)



P (2)

- P is the set of all decision problems solvable in polynomial time on **REAL** computers.



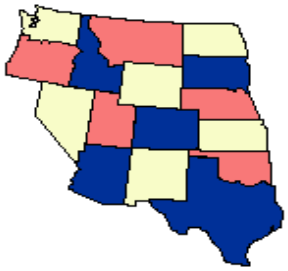
Short Certificates

- To find a solution for an NPC problem, we seem to be required to try out exponential amounts of partial solutions
- Failing in extending a partial solution requires **backtracking**
- However, once we found a solution, convincing someone of it is easy, if we keep a proof, i.e., a **certificate**
- The problem is finding an answer (exponential), but not verifying a potential solution (polynomial)

Short Certificates (2)

Input x:

Certificate c:



Verifier:
1. Check that x and c describe same map.
2. Count number of distinct colors in c.
3. Check all pairs of adjacent states.

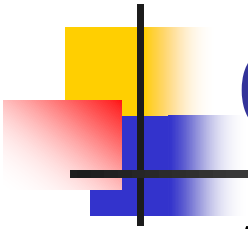
YES

NO

3-COLOR is in NP.

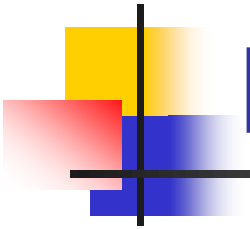
x is a YES instance

no conclusion



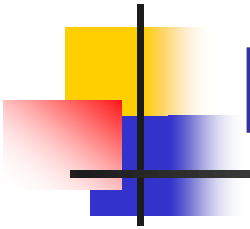
On Magic Coins and Oracles

- Assume we use a magic coin in the backtracking algorithm
  - whenever it is possible to extend a partial solutions in ”two” ways, we perform a coin toss (two monkey cards, next truth assignment, etc.)
  - the outcome of this ”act” determines further actions – we use magical insight, supernatural powers!
- Such algorithms are termed ”**non-deterministic**”
  - they **guess** which option is better, **rather** than employing some **deterministic procedure** to go through the alternatives



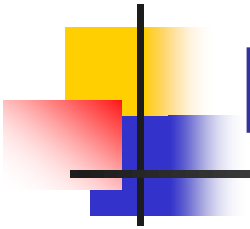
NP

- Definition of NP:
  - Set of all decision problems solvable in polynomial time on a NONDETERMINISTIC Turing machine
  - Definition important because it links many fundamental problems
- Useful alternative definition
  - Set of all decision problems with efficient verification algorithms
    - efficient = polynomial number of steps on deterministic TM
  - Verifier: algorithm for decision problem with extra input



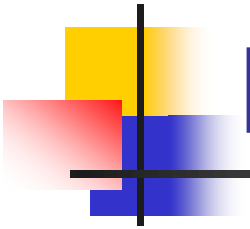
NP (2)

- NP = set of decision problems with efficient verification algorithms
- Why doesn't this imply that all problems in NP can be solved efficiently?
  - BIG PROBLEM: need to know certificate ahead of time
    - real computers can simulate by guessing all possible certificates and verifying
    - naïve simulation takes exponential time unless you get "lucky"



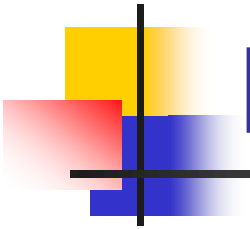
NP-Completeness

- Informal definition of NP-hard:
  - A problem with the property that if it can be solved efficiently, then it can be used as a subroutine to solve any other problem in NP efficiently
- NP-complete problems are NP problems that are NP-hard
  - ”Hardest computational problems” in NP



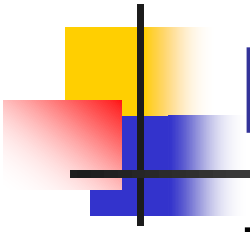
NP-Completeness (2)

- Each NPC problem's faith is tightly coupled to all the others (complete set of problems)
- Finding a **polynomial time algorithm for one NPC problem** would **automatically** yield an a polynomial time algorithm **for all NP problems**
- Proving that one NP-complete problem has an **exponential lower bound** woud **automatically** proove that **all other NP-complete** problems have exponential lower bounds



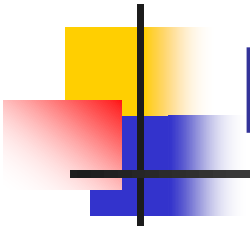
NP-Completeness (3)

- *How can we prove such a statement?*
- Polynomial time reduction!
  - given two problems
  - it is an algorithm running in polynomial time that reduces one problem to the other such that
    - given input X to the first and asking for a yes/no answer
    - we transform X into input Y to the second problem such that its answer matches the answer of the first problem



Reduction Example

- Reduction is a general technique for showing that one problem is harder (easier) than another
  - For problems A and B, we can often show: if A can be solved efficiently, then so can B
  - In this case, we say B reduces to A (B is "easier" than A, or, B cannot be ”worse” than A)

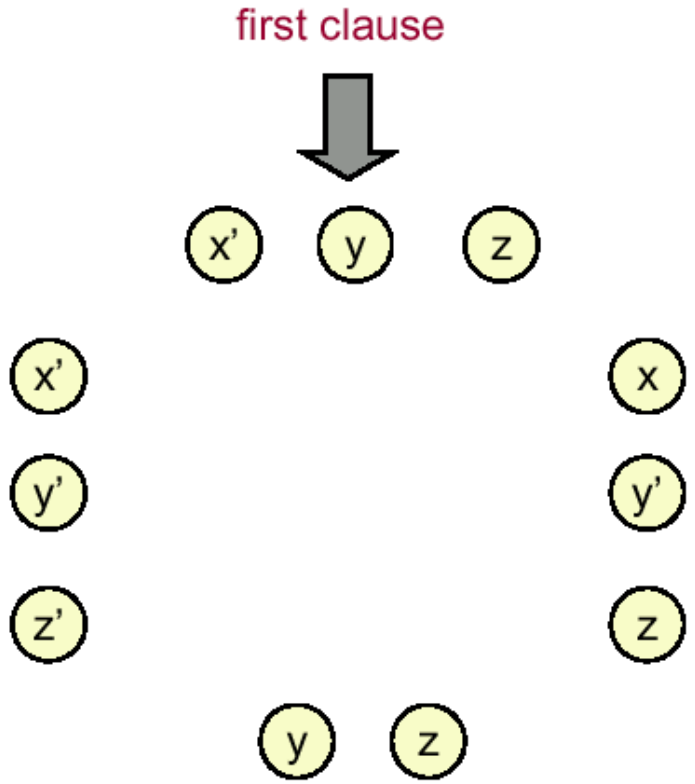


Redcution Example (2)

- SAT reduces to CLIQUE
  - Given any input to SAT, we create a corresponding input to CLIQUE that will help us solve the original SAT problem
  - Specifically, for a SAT formula with K clauses, we construct a CLIQUE input that has a clique of size K if and only if the original Boolean formula is satisfiable
  - If we had an efficient algorithm for CLIQUE, we could apply our transformation, solve the associated CLIQUE problem, and obtain the yes- no answer for the original SAT problem

Reduction Example (3)

- SAT reduces to CLIQUE
  - Associate a person to each variable occurrence in each clause



Boolean formula:

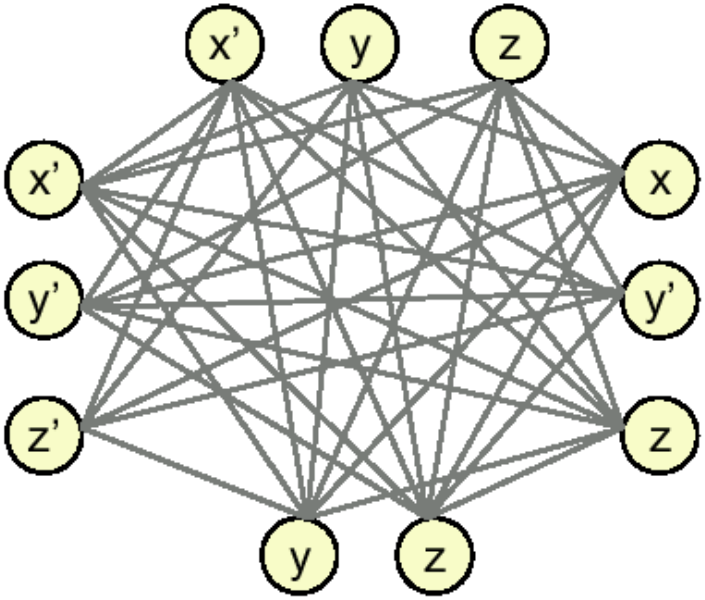
(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')

Reduction Example (4)

- SAT reduces to CLIQUE
  - Associate a person to each variable occurrence in each clause
  - ”Two people” know each other except if:
    - they come from the same clause
    - they represent t and t’ for some variable t

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')

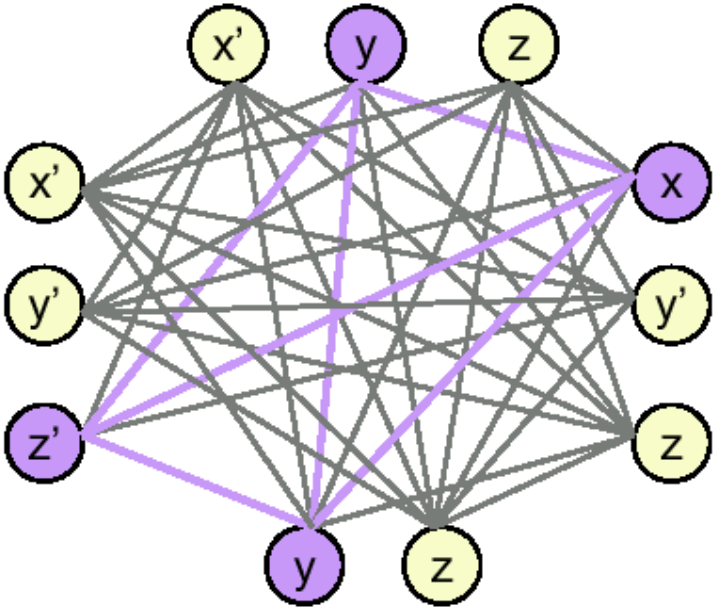


Reduction Example (5)

- SAT reduces to CLIQUE
  - Two people know each other except if:
    - they come from the same clause
    - they represent t and t' for some variable t
  - Clique of size 4 $\Rightarrow$ satisfiable assignment
    - set variable in clique to "true"
    - (x, y, z) = (true, true, false)

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')

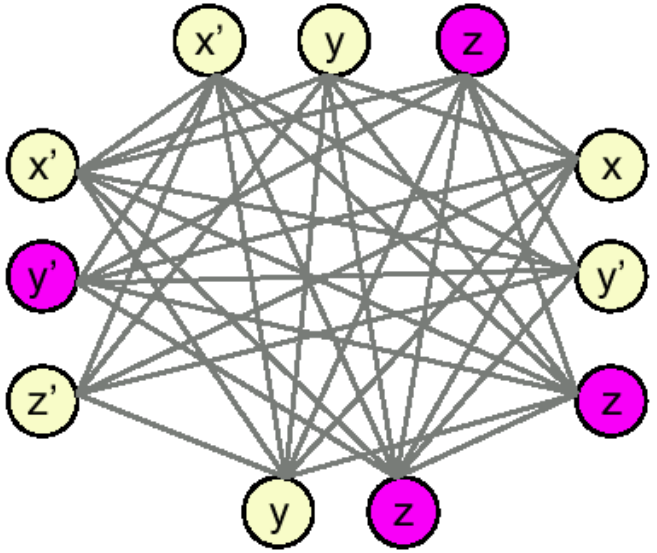


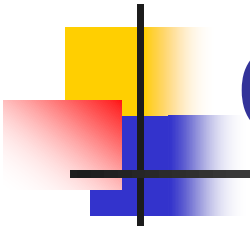
Reduction Example (6)

- SAT reduces to CLIQUE
  - Two people know each other except if:
    - they come from the same clause
    - they represent t and t’ for some variable t
  - Clique of size 4 $\Rightarrow$ satisfiable assignment
  - Satisfiable assignment $\Rightarrow$ clique of size 4
    - (x, y, z) = (false, false, true)
    - choose one true literal from each clause

Boolean formula:

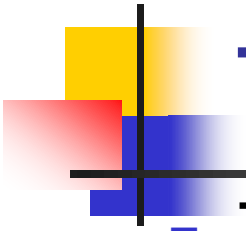
(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')





CLIQUE is NP-complete

- CLIQUE is NP-complete
  - CLIQUE is in NP
  - SAT is in NP-complete
  - SAT reduces to CLIQUE
- Hundreds of problems can be shown to be NP-complete that way…

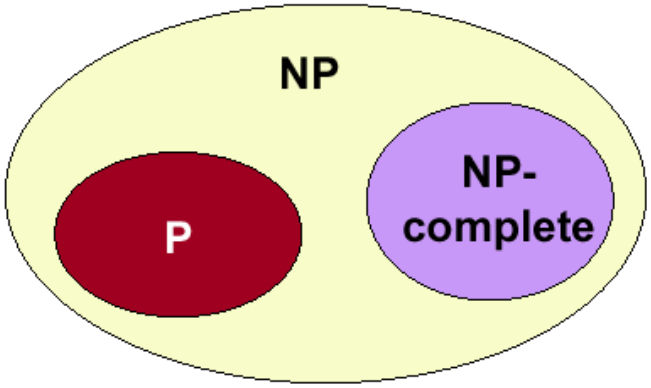


The Start…

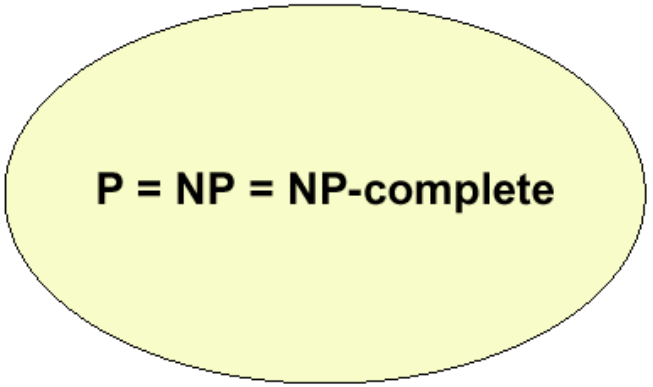
- The World's first NP-complete problem
- SAT is NP- complete. (Cook- Levin, 1960's)
- Idea of proof:
  - By definition, nondeterministic TM can solve problem in NP in polynomial time
  - Polynomial- size Boolean formula can describe (nondeterministic) TM
  - Given any problem in NP, establish a correspondence with some instance of SAT
  - SAT solution gives simulation of TM solving the corresponding problem
  - IF SAT can be solved in polynomial time, then so can any problem in NP (e. g., TSP).

The Main Question

- Does P = NP? (Edmonds, 1962)
  - Is the original DECISION problem as easy as VERIFICATION?
- Most important open problem in theoretical computer science. Clay institute of mathematics offers one-million dolar prize!



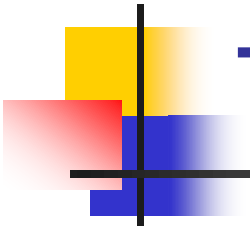
If $P \neq NP$



If P = NP

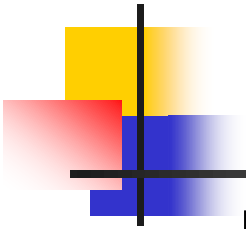
The Main Question (2)

- If P=NP, then:
  - Efficient algorithms for 3- COLOR, TSP, and factoring.
  - Cryptography is impossible on conventional machines
  - Modern banking system will collapse
- If no, then:
  - Can't hope to write efficient algorithm for TSP
    - see NP- completeness
  - But maybe efficient algorithm still exists for testing the primality of a number – i.e., there are some problems that are NP, but not NP-complete



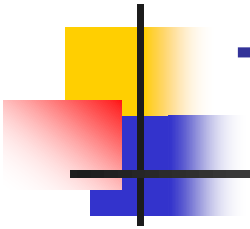
The Main Question (3)

- Probably no, since:
  - Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP-complete problems without success
  - Consensus opinion: $P \neq NP$
- But maybe yes, since:
  - No success in proving $P \neq NP$ either



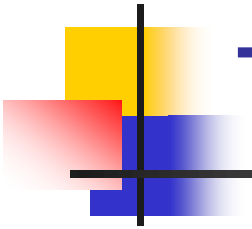
Dealing with NP-Completeness

- Hope that a worst case doesn't occur
  - Complexity theory deals with worst case behavior. The instance(s) you want to solve may be "easy"
    - TSP where all points are on a line or circle
    - 13,509 US city TSP problem solved (Cook et. al., 1998)
- Change the problem
  - Develop a heuristic, and hope it produces a good solution.
  - Design an approximation algorithm: algorithm that is guaranteed to find a high- quality solution in polynomial time
    - active area of research, but not always possible
- Keep trying to prove P = NP.



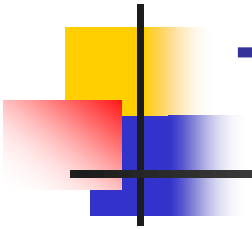
The Big Picture

- Summarizing: it is not known whether NP problems are tractable or intractable
- But, there exist provably intractable problems
  - Even worse – there exist problems with running times unimaginably worse than exponential!
- More bad news: there are **provably noncomputable (undecidable)** problems
  - There are no (and there will not ever be!!!) algorithms to solve these problems



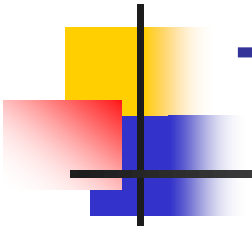
The Course

- Algorithmic problems and solutions
- Correctness of algorithms
- Asymptotic notations
- Recursion/recurrences
- Algorithmic techniques
  - Divide and Conquer (Merge sort, Quicksort, Binary search, Closest pair)
  - Dynamic programming (Matrix chain multiplication, Longest Common Subsequence)
  - Greedy algorithms (Prim's, Kruskal's, Dijkstra's)



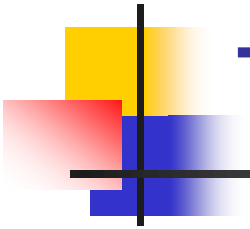
The Course (2)

- Sorting
  - insertion sort
  - merge sort
  - quick sort
  - heap sort (priority queues)
- Simple data structures (array, all sorts of linked lists, stack, queues, trees, heaps)
- Dictionaries (fast data access)
  - binary trees (unbalanced)
  - red-black trees
  - B-trees



The Course (3)

- Graphs
  - what is it?
  - graph traversal
    - breadth-first search
    - depth-first search (topological sort)
  - minimum spanning trees (Prim, Kruskal)
  - shortest path (Dijkstra, Bellman-Ford)
- Computational Geometry
  - what is it all about (points and lines)
  - sweep line
  - line segment intersections
  - closest pairs



The Course (4)

- Complexity classes
  - what's good and what's not
  - NP-completeness
  - reducibility and examples