

Two one million US dollar bills are shown, tilted diagonally. The top bill is a 1993 series bill featuring the Statue of Liberty and the text 'U.S. MILLENNIUM NOTE'. The bottom bill is a 1934 series bill featuring the US Capitol building. Both bills are green and have 'ONE MILLION DOLLARS' printed on them.

The $P = NP?$ -question

Ulrike Stege

LAST CLASS ...

- Graph Algorithms for
 - Minimum Spanning Tree
 - Small Parsimony Problem
- Both algorithms run in polynomial time with respect to the input size of the graph.

ANOTHER COMPUTATIONAL PROBLEM THAT CAN BE SOLVED USING A GRAPH ALGORITHM

Topological Sort

- Input: A set of n tasks that have to be done; a set of m constraints on the tasks, indicating that certain tasks have to be performed before certain other tasks
- Output: An order of performing all the tasks such that every constraint is satisfied (if such an order exists)

TOPOLOGICAL SORT HAS APPLICATIONS IN

- Project Management
- Scheduling
- Ordering of formula cell-evaluation when computing formula values in spreadsheets
- Determining the order of compilation tasks to perform in makefiles

HOW CAN WE SOLVE TOPOLOGICAL SORT?

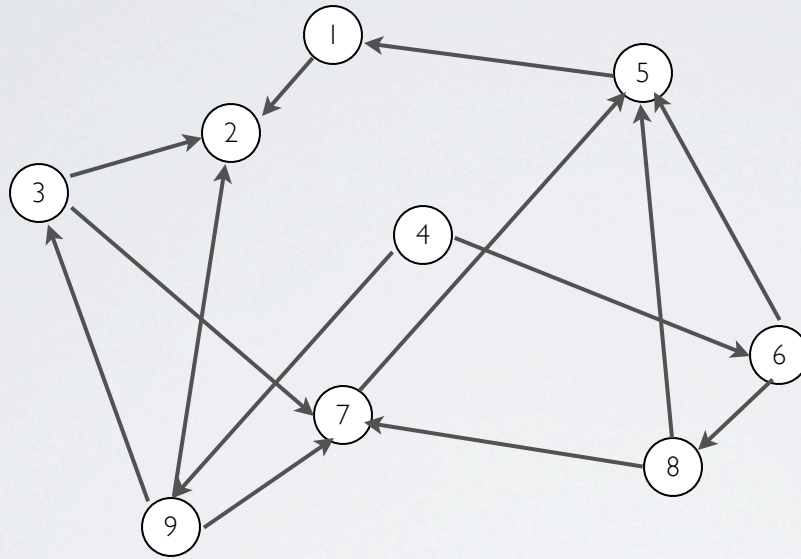
Example: baking tasks

1. Preheat oven to 190 degrees C. (1)
2. Bake for 15 minutes. (2)
3. Beat to mix. (3)
4. In a bowl, beat butter until fluffy. (4)
5. With a scoop produce evenly sized cookies and place on cookie sheet. (5)
6. Add flour to butter mixture and stir until mixed. (6)
7. Add nuts to cookie mixture. (7)
8. Slowly add oil and beat until well incorporated. (8)
9. Add egg and vanilla. (9)

Constraints

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

HOW CAN WE SOLVE TOPOLOGICAL SORT?



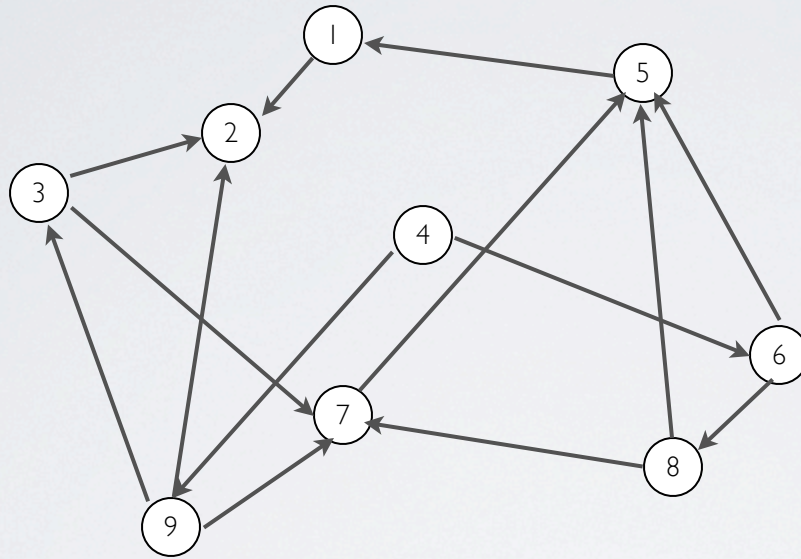
Constraints

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

Legend: (1) Tasks

→ Constraints

HOW CAN WE SOLVE TOPOLOGICAL SORT?



Constraints

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

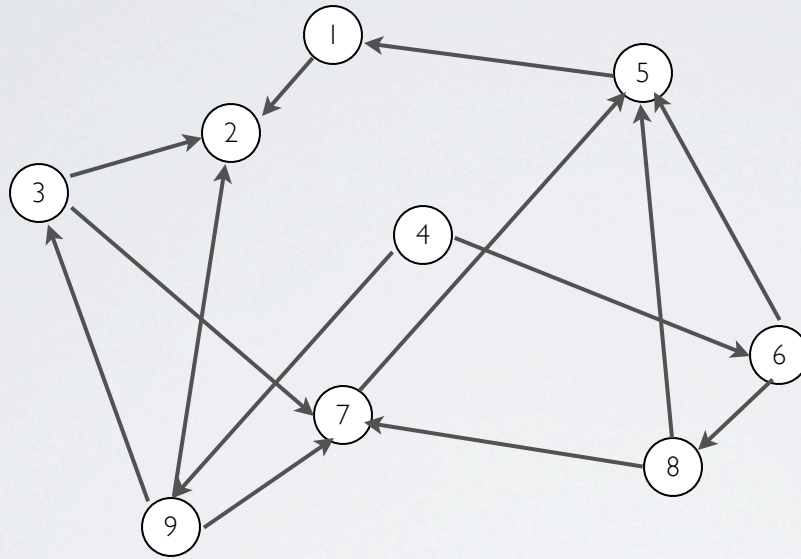
4 6 8 9 3 7 5 1 2

- h 1. Preheat oven to 190 degrees C.
- i 2. Bake for 15 minutes.
- e 3. Beat to mix.
- a 4. In a bowl, beat butter until fluffy.
- g 5. With a scoop produce evenly sized cookies and place on cookie sheet.
- b 6. Add flour to butter mixture and stir until mixed.
- f 7. Add nuts to cookie mixture.
- C 8. Slowly add oil and beat until well incorporated.
- d 9. Add egg and vanilla.

4 6 8 9 3 7 5 1 2

- a. In a bowl, beat butter until fluffy.
- b. Add flour to butter mixture and stir until mixed.
- c. Slowly add oil and beat until well incorporated.
- d. Add egg and vanilla.
- e. Beat to mix.
- f. Add nuts to cookie mixture.
- g. With a scoop produce evenly sized cookies and place on cookie sheet.
- h. Preheat oven to 190 degrees C.
- i. Bake for 15 minutes

TOPOLOGICAL SORT ALGORITHM

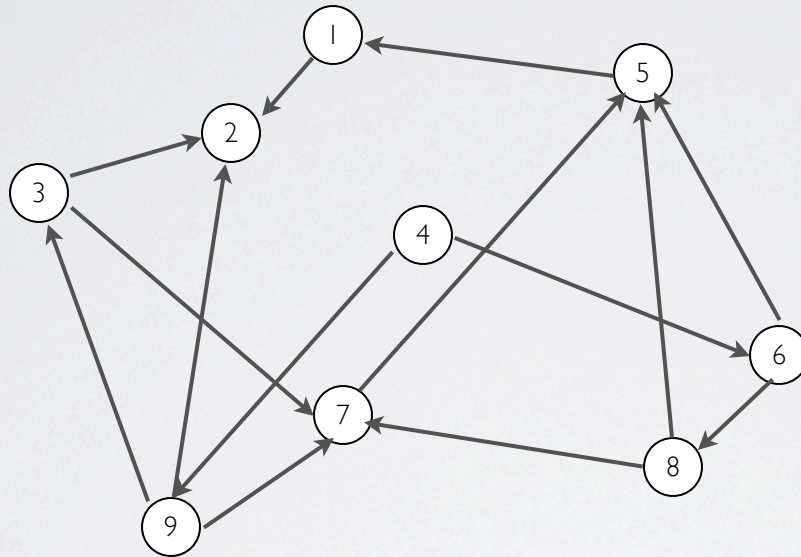


*Does this
work for any
input of tasks
and
constraints?*

while there are unpicked tasks **do**

pick a task that has no incoming constraints

TOPOLOGICAL SORT ALGORITHM



while there are unpicked tasks & there exists a task that has no incoming constraints **do**

 pick such a task that has no incoming constraints

If there is a task that is not picked **then**
 output “Mission impossible”

TOPOLOGICAL SORT

- How many operations does the algorithm perform for n given tasks?
- To pick a task, we need to check all tasks left to see if there is one that has no incoming constraint.
- Once we found and selected one, we need to remove the outgoing constraints from the selected task.
(There can be as many outgoing constraints at the selected tasks as tasks left!)
- These two steps have to be repeated until no task can be picked anymore.

Topological Sort can be solved in *quadratic time* in n or $O(n^2)$.

POLYNOMIAL (RUNNING) TIME

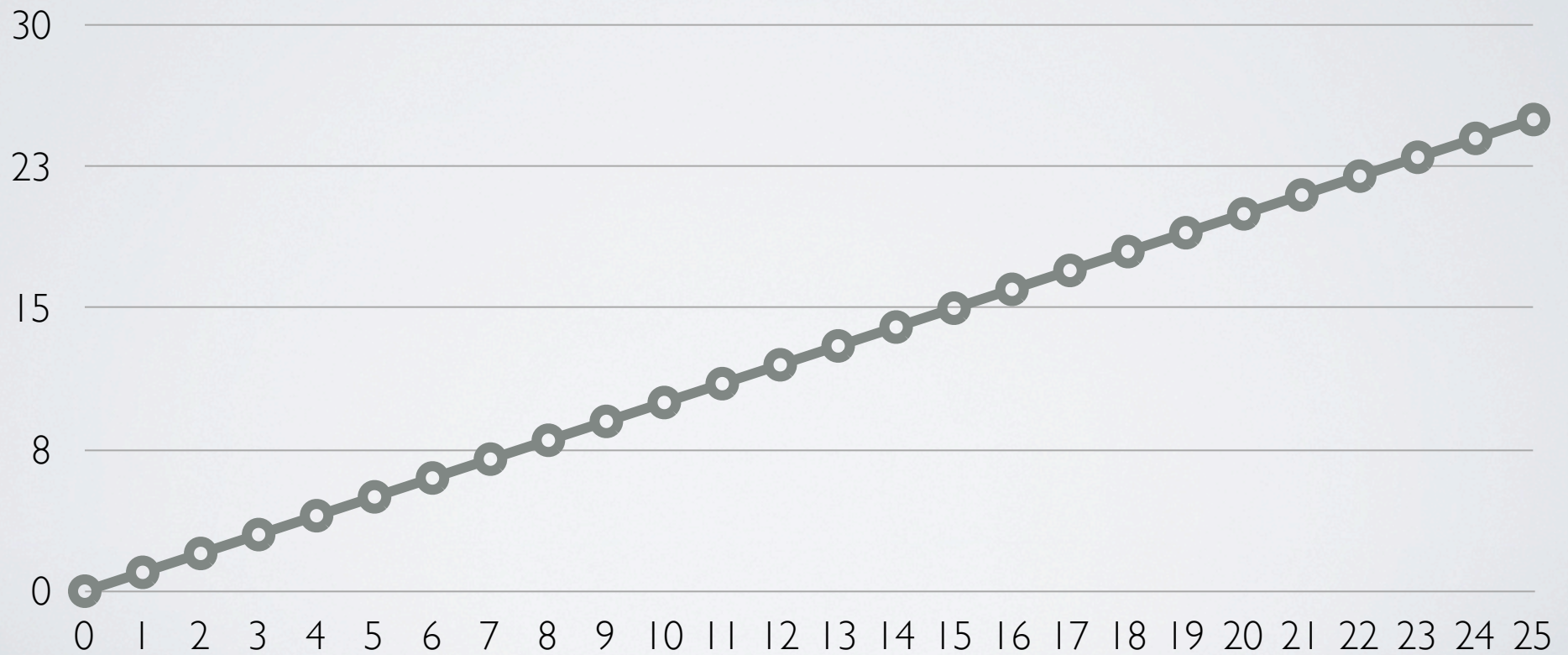
- For a computational problem with input size n , an algorithm solving the problem runs in *polynomial time* if it uses in the worst case at most n^c operations.
- Here, c is a constant (Example: $c = 2$ for a quadratic time).

THE COMPLEXITY CLASS P

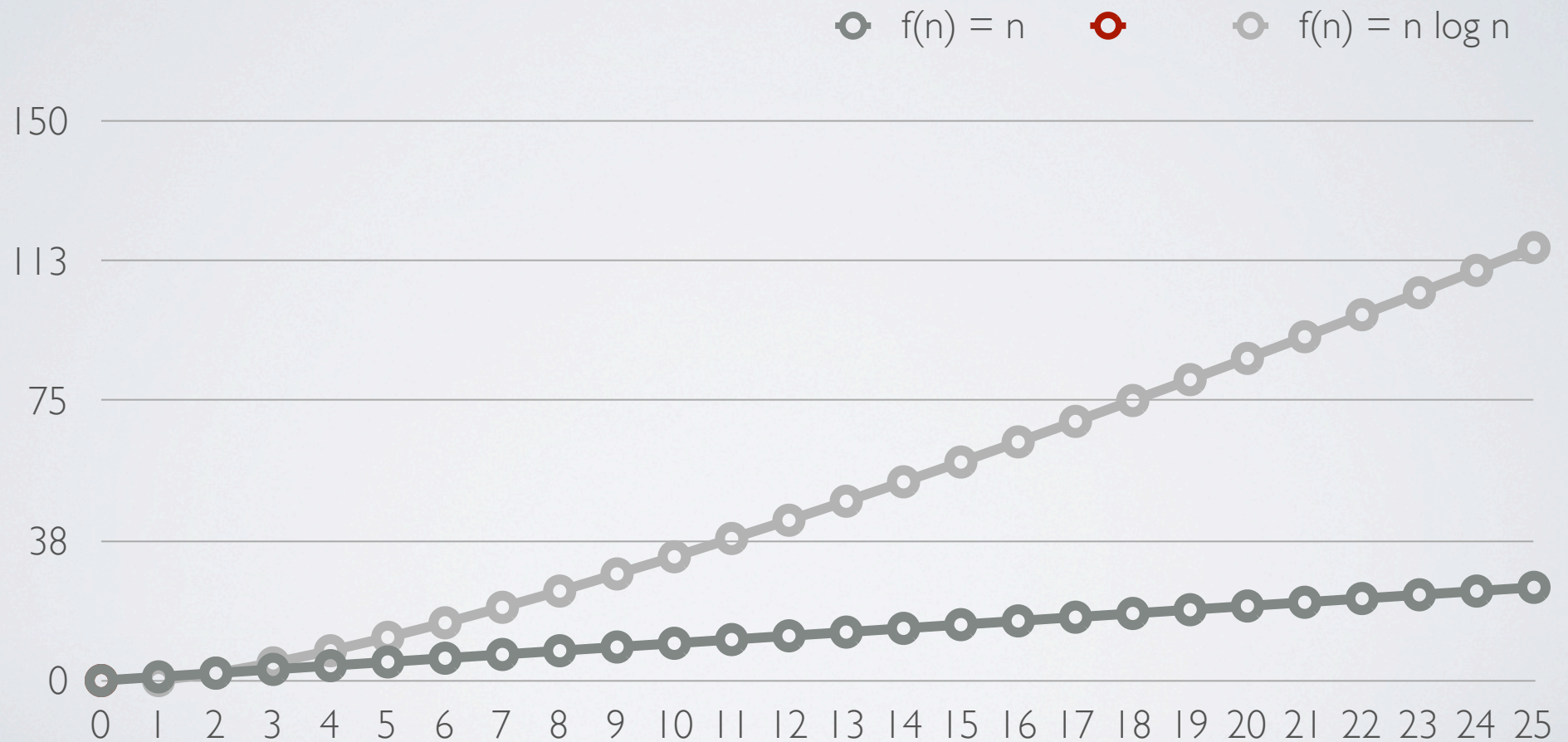
- The *class P* contains all the computational problems that can be solved (by an algorithm) in polynomial time.
- The computational problems Sorting, Search, Minimum Spanning Tree, Topological Sort, and Small Parsimony are all members of the class P.

POLYNOMIAL GROWTH RATES

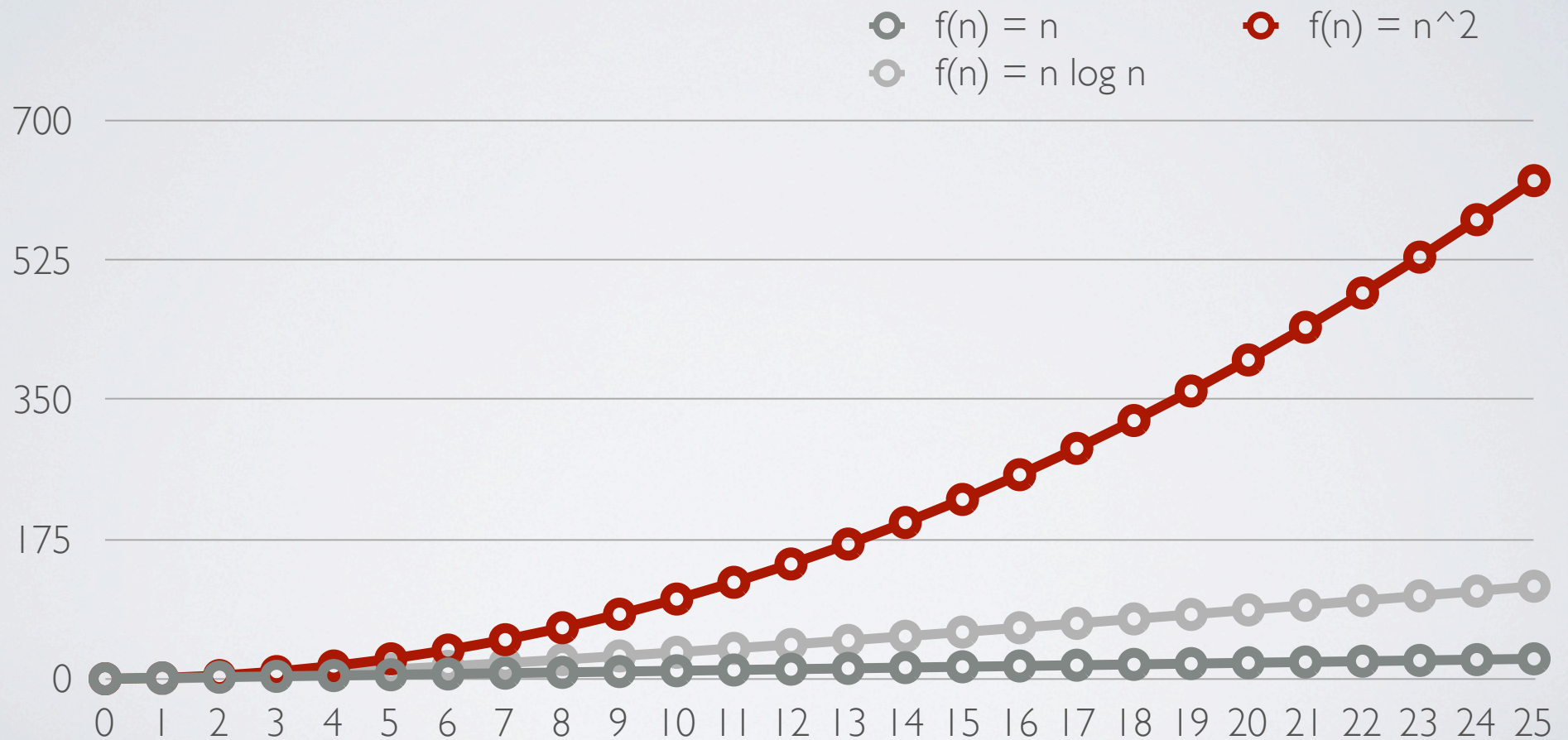
○ $f(x) = x$



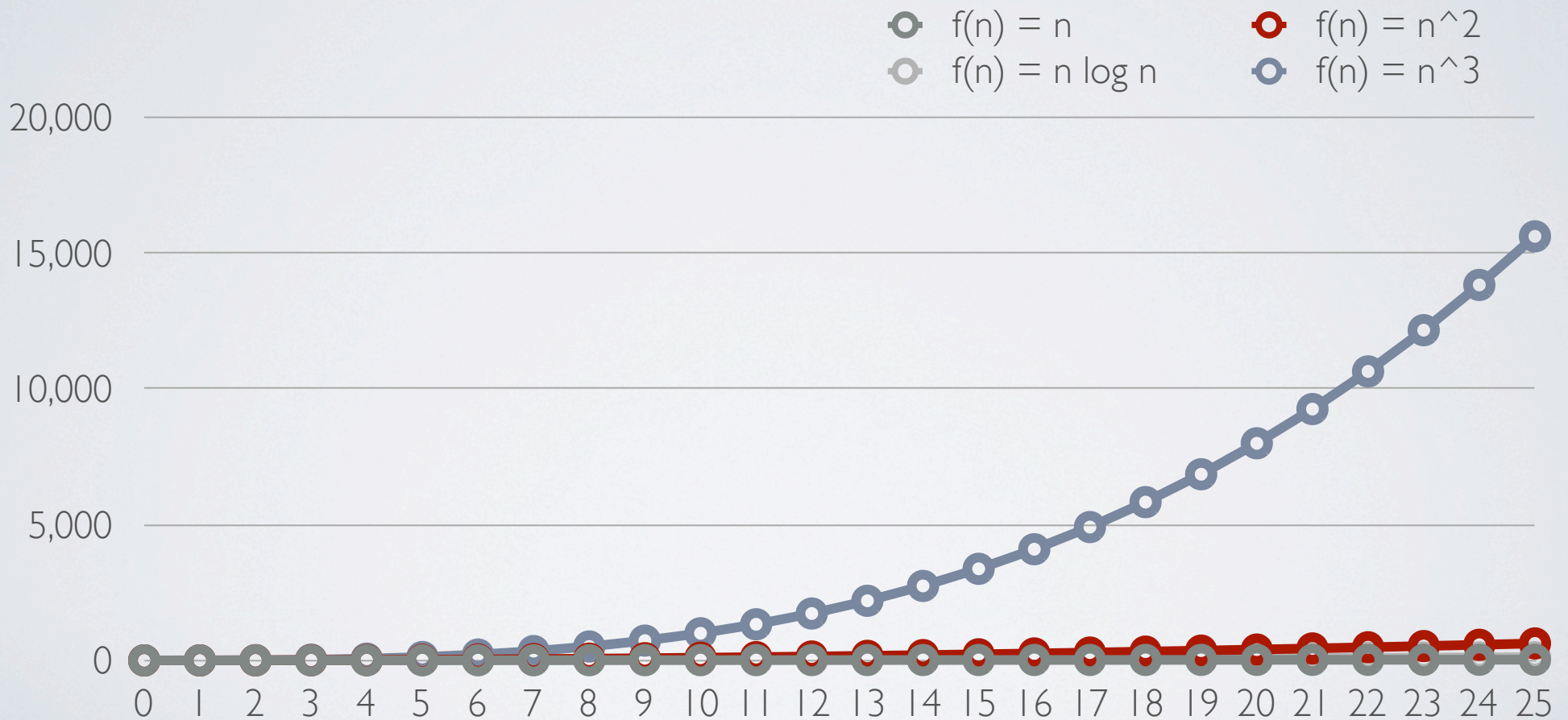
POLYNOMIAL GROWTH RATES



POLYNOMIAL GROWTH RATES



POLYNOMIAL GROWTH RATES



A SCHEDULING PROBLEM

- Given is a list of all the first-year courses, including their lecture times, that you have the prerequisites for
- You are to enter **as many courses as possible** into your schedule but
 - courses cannot conflict in lecture times

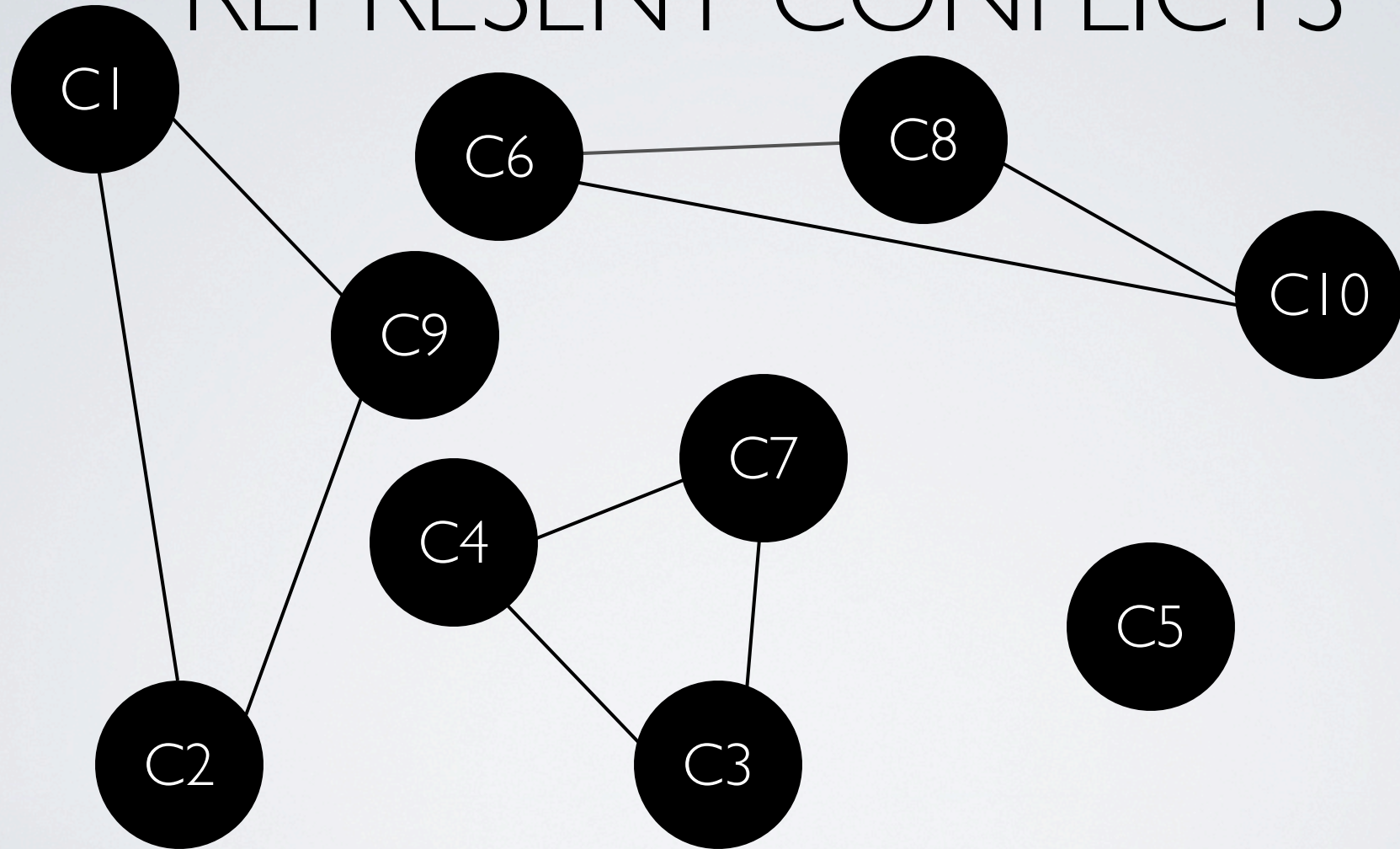
SCHEDULING

- C1 MR 1-2:30
 - C2 MWF 1-2
 - C3 MT 8-10:30
 - C4 TWF 9-10
 - C5 MTW 4-5
 - C6 TRF 12-1
 - C7 T 9-12
 - C8 WF 11-12:30
 - C9 TWR 1-2
 - C10 RF 12-1
-
- The diagram illustrates scheduling conflicts between 10 courses. Red lines connect courses that share overlapping time slots:
- C1 (MR 1-2:30) conflicts with C2 (MWF 1-2), C3 (MT 8-10:30), C4 (TWF 9-10), C7 (T 9-12), and C8 (WF 11-12:30).
 - C2 (MWF 1-2) conflicts with C1 (MR 1-2:30), C3 (MT 8-10:30), C4 (TWF 9-10), C7 (T 9-12), and C8 (WF 11-12:30).
 - C3 (MT 8-10:30) conflicts with C1 (MR 1-2:30), C2 (MWF 1-2), C4 (TWF 9-10), C5 (MTW 4-5), C6 (TRF 12-1), C7 (T 9-12), and C8 (WF 11-12:30).
 - C4 (TWF 9-10) conflicts with C1 (MR 1-2:30), C2 (MWF 1-2), C3 (MT 8-10:30), C5 (MTW 4-5), C6 (TRF 12-1), C7 (T 9-12), and C8 (WF 11-12:30).
 - C5 (MTW 4-5) conflicts with C3 (MT 8-10:30), C4 (TWF 9-10), and C6 (TRF 12-1).
 - C6 (TRF 12-1) conflicts with C3 (MT 8-10:30), C4 (TWF 9-10), C5 (MTW 4-5), C7 (T 9-12), C8 (WF 11-12:30), and C10 (RF 12-1).
 - C7 (T 9-12) conflicts with C1 (MR 1-2:30), C2 (MWF 1-2), C3 (MT 8-10:30), C4 (TWF 9-10), C6 (TRF 12-1), and C8 (WF 11-12:30).
 - C8 (WF 11-12:30) conflicts with C1 (MR 1-2:30), C2 (MWF 1-2), C3 (MT 8-10:30), C4 (TWF 9-10), C6 (TRF 12-1), C7 (T 9-12), and C10 (RF 12-1).
 - C9 (TWR 1-2) has no conflicts shown.
 - C10 (RF 12-1) conflicts with C6 (TRF 12-1) and C8 (WF 11-12:30).

WHAT INFORMATION IN THE INPUT IS IMPORTANT?

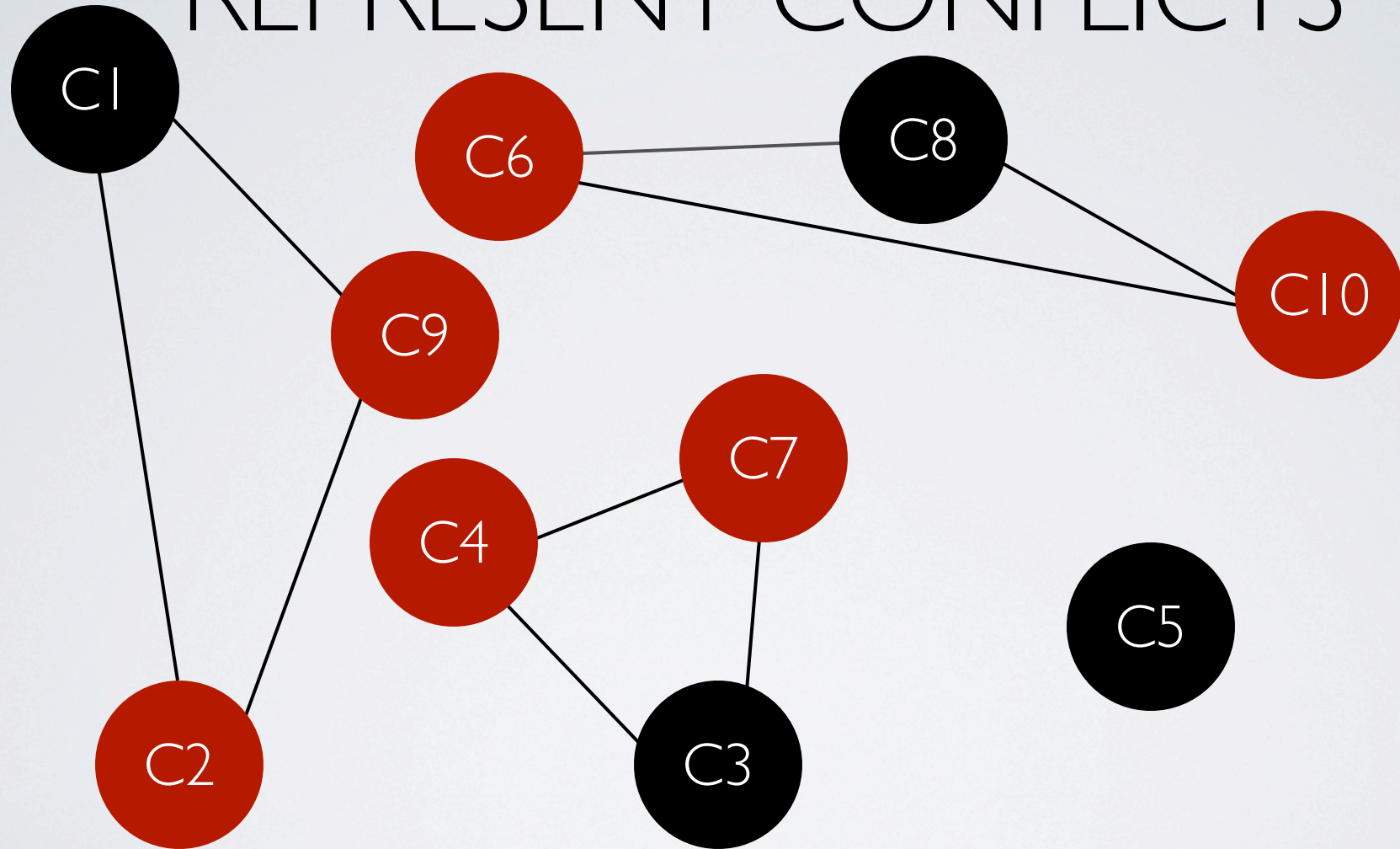
- We identified the pairs of courses that *conflict*.
- We obtain a *conflict graph* that contains exactly that information but suppresses the specific lecture times.

CONFLICT **GRAPH**: EDGES REPRESENT CONFLICTS



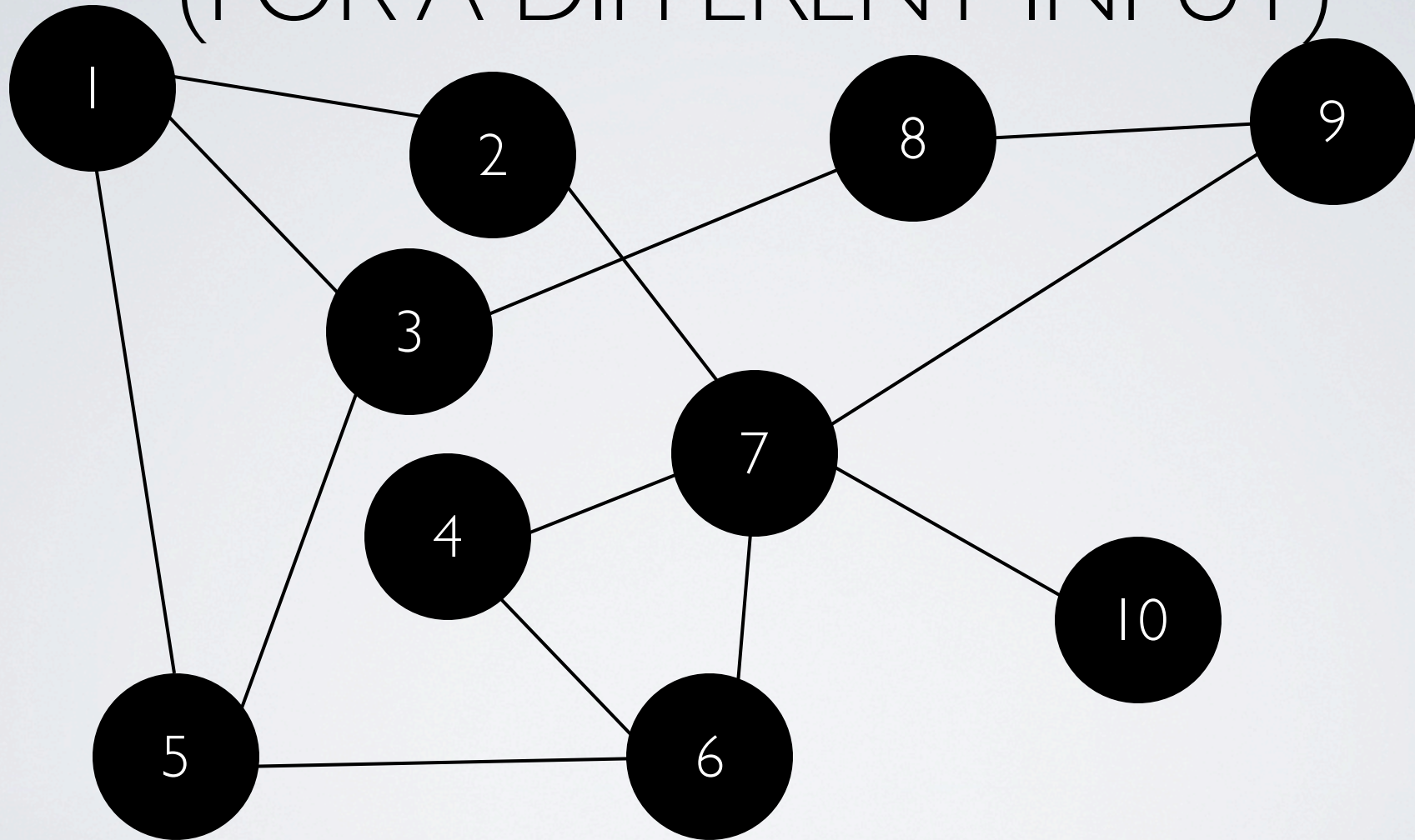
Goal: Remove conflicting courses such that a maximum number of courses remains!

CONFLICT **GRAPH**: EDGES REPRESENT CONFLICTS



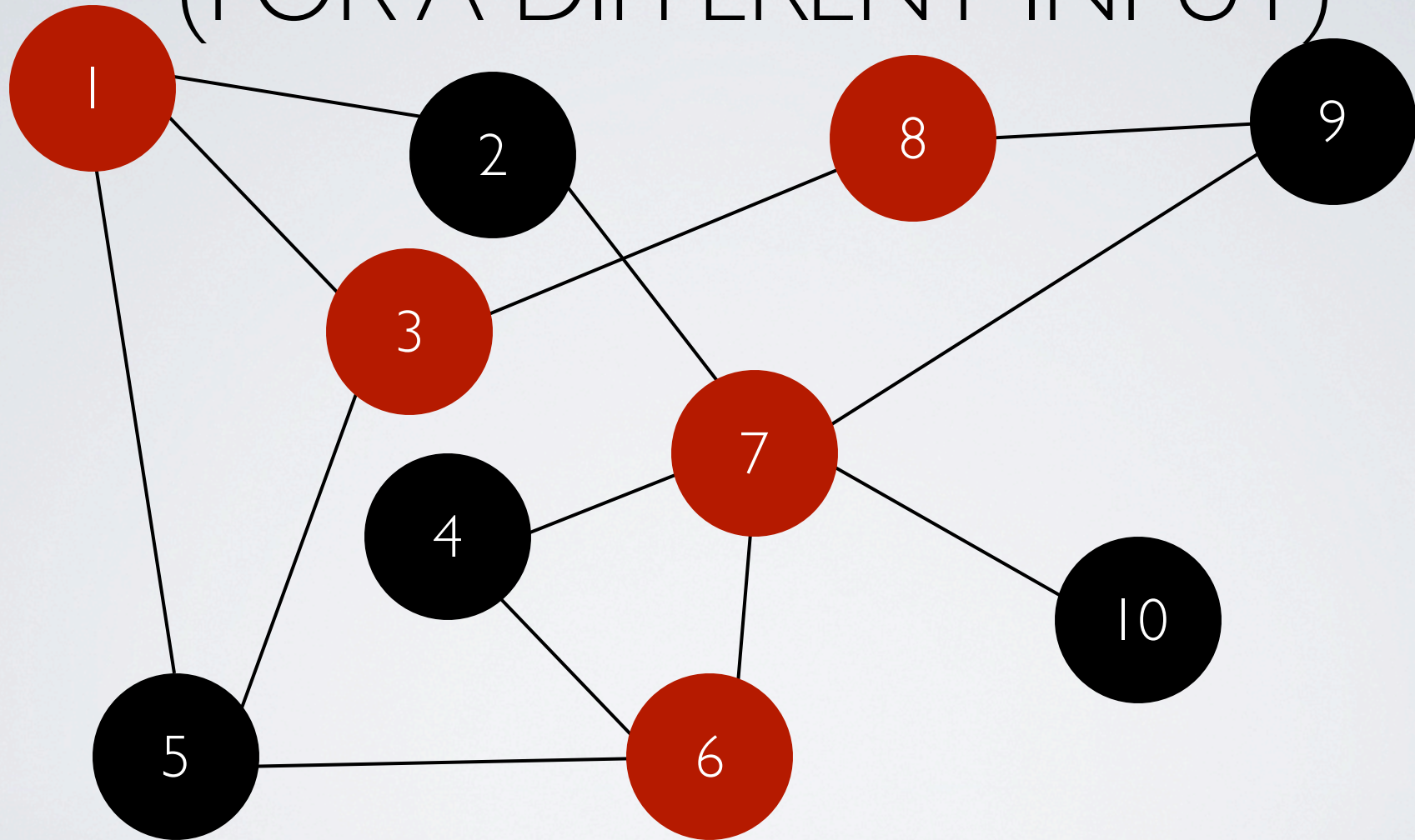
Goal: Remove conflicting courses such that a maximum number of courses remains!

A DIFFERENT CONFLICT GRAPH (FOR A DIFFERENT INPUT)



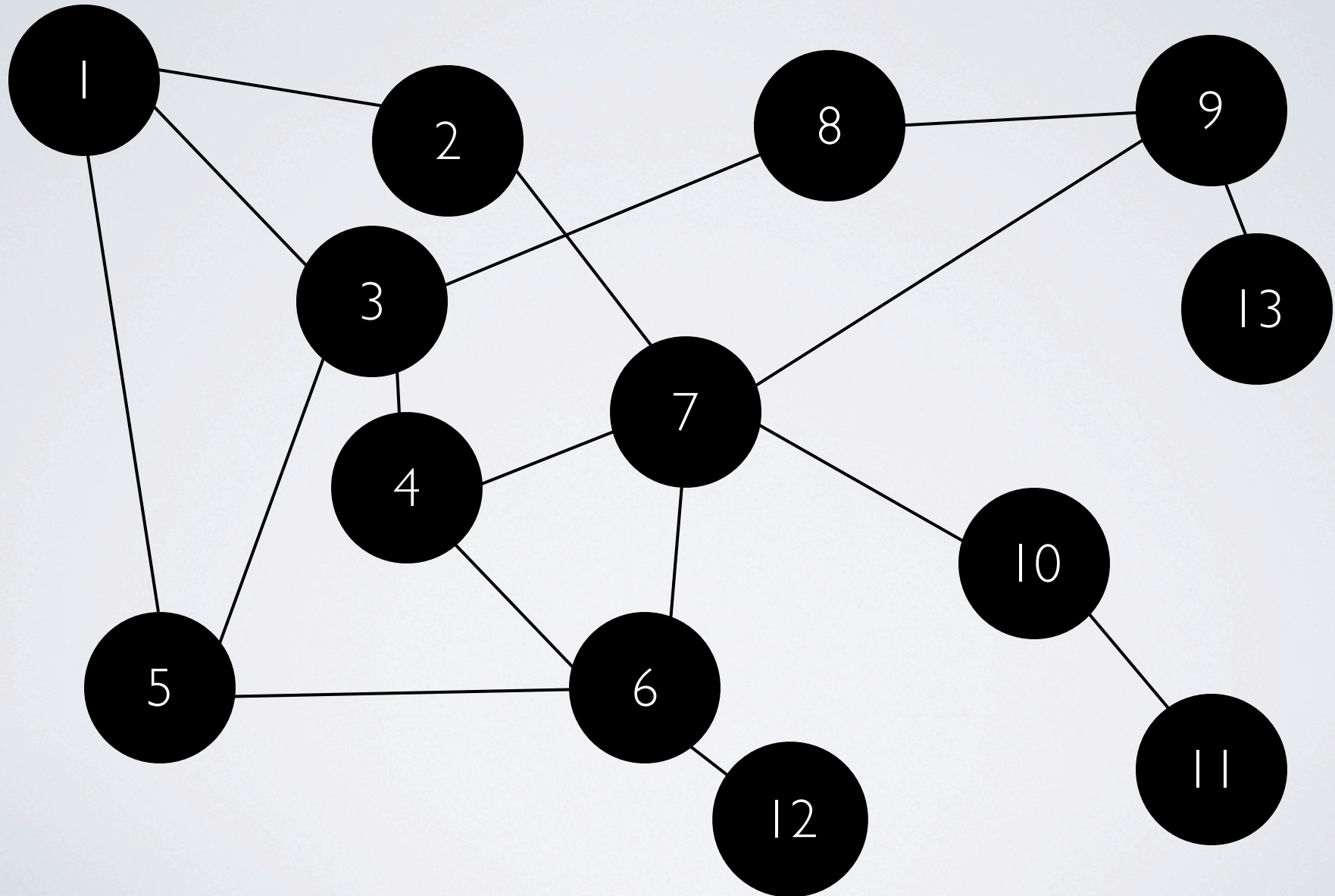
Remove conflicting vertices such that a maximum number of vertices remains!

A DIFFERENT CONFLICT GRAPH (FOR A DIFFERENT INPUT)

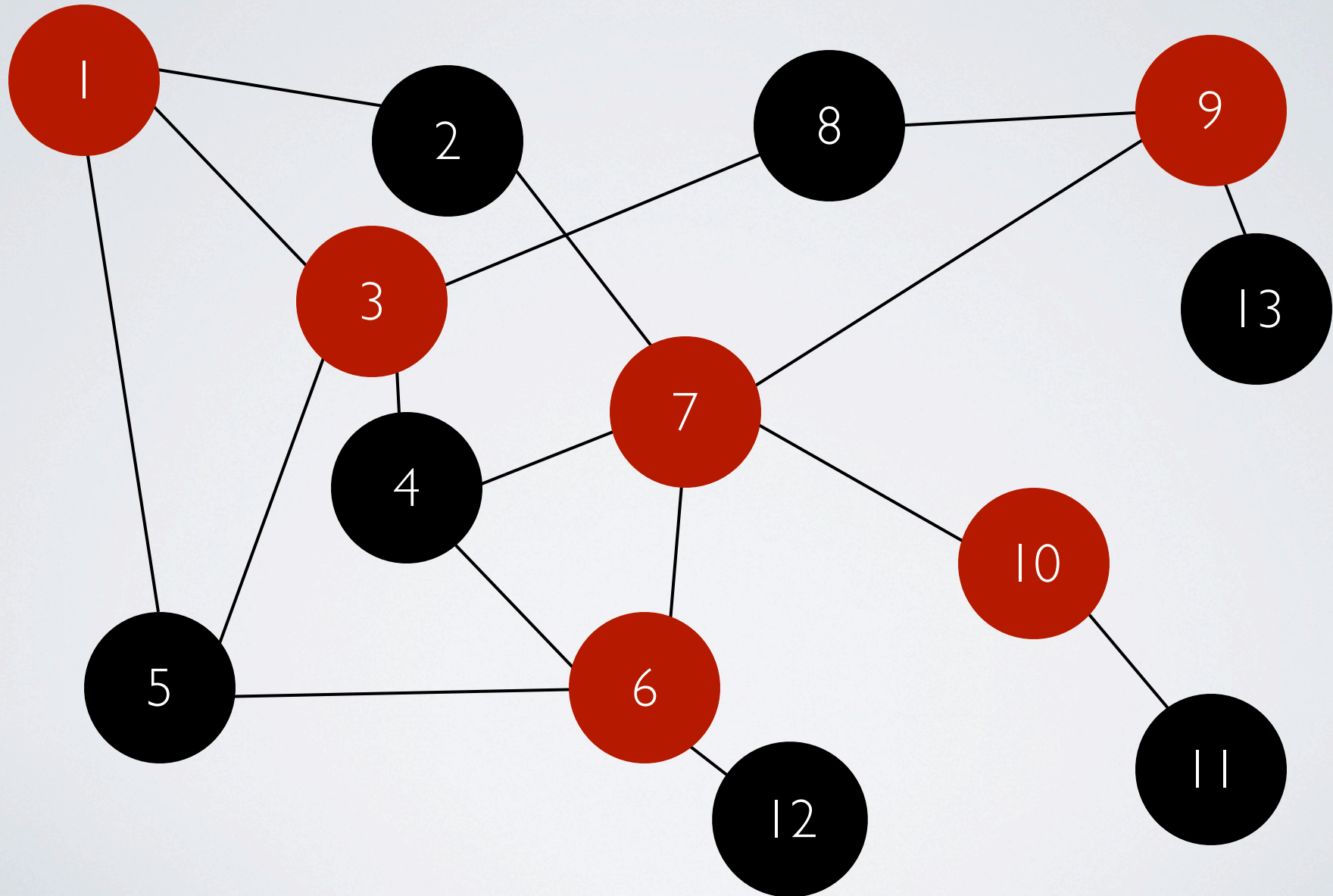


Remove conflicting vertices such that a maximum number of vertices remains!

ANOTHER ONE



ANOTHER ONE



WHAT ALGORITHM DOES SOLVE THIS SCHEDULING PROBLEM?

- **Step 1:** For each possible subset/combination of vertices/courses do the following:
 - check whether removing all the other vertices/courses results in a conflict-free graph
- **Step 2:** Of all those combinations (or *candidate solutions*) that are conflict free, determine one that is largest

This can be done in 2^n many steps/operations.

SUPER-POLYNOMIAL RUNNING TIMES

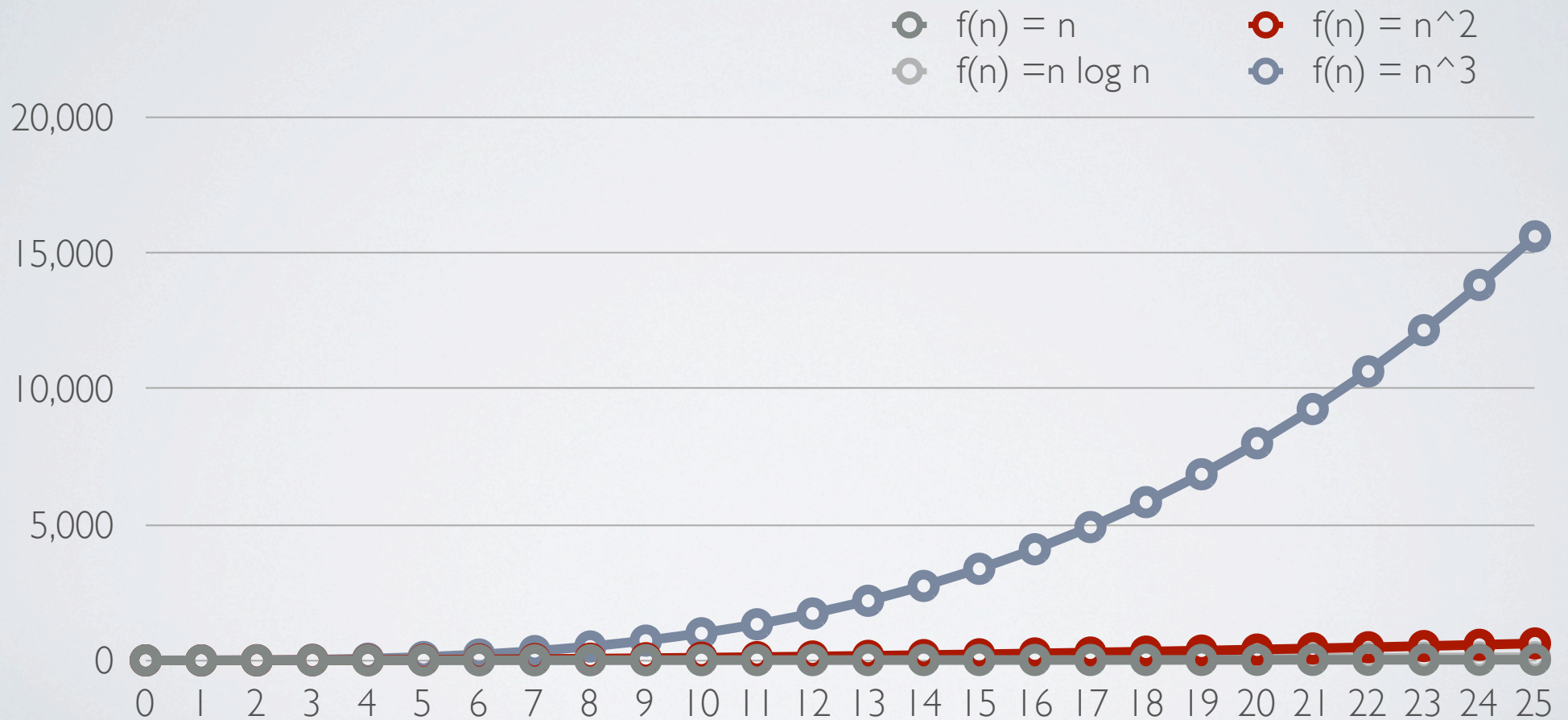
- The number of steps is not bounded by n^c (c is a constant, n is the input size)
- Example: running times of the form c^n (c is a constant, n is the input size), such as 2^n , 3^n , 4^n , 5^n

POLYNOMIAL TIMES VS. EXPONENTIAL TIMES

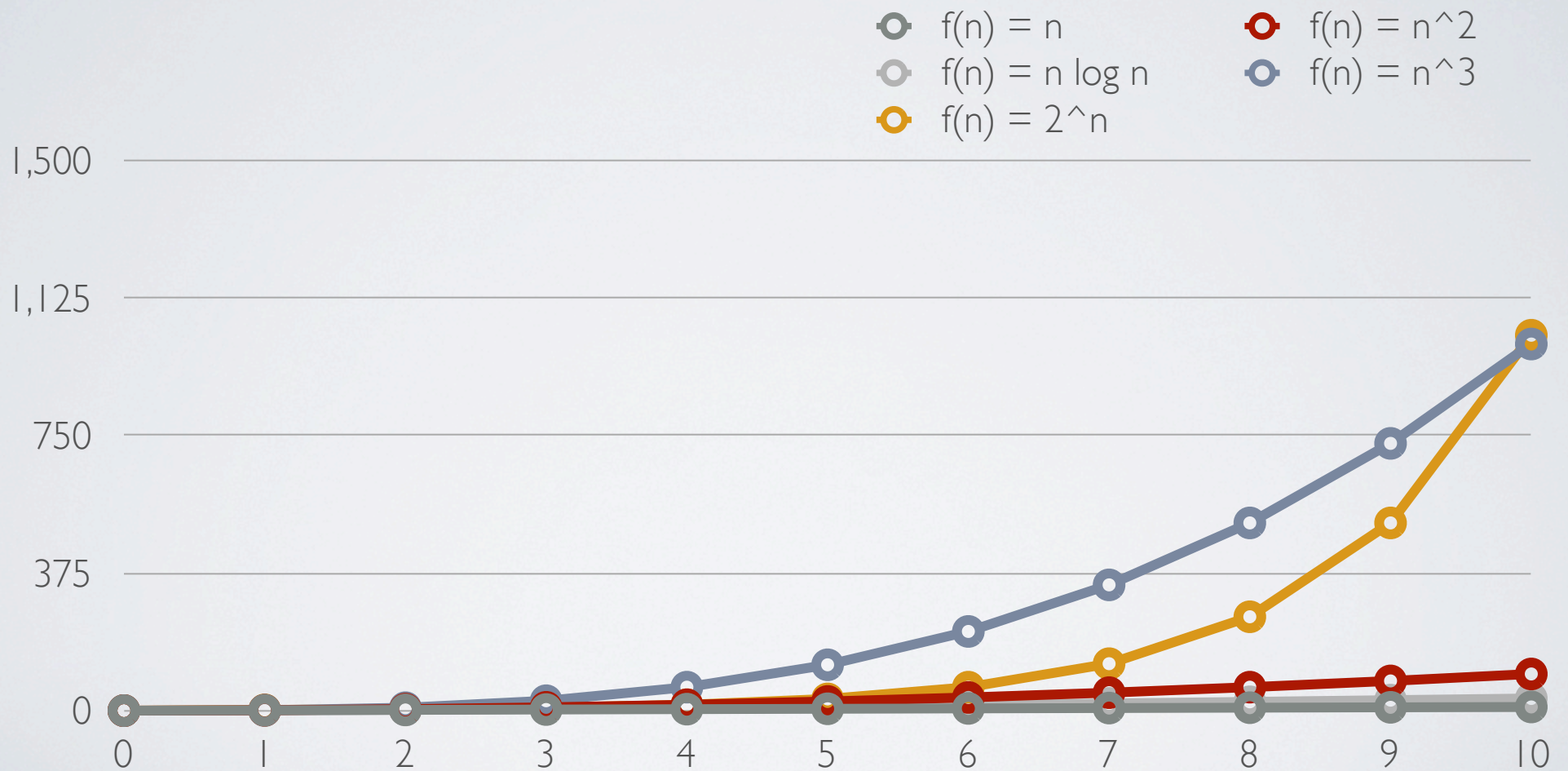
input size n	n^2	2^n
5	0.25 sec	0.32 sec
10	1 sec	10.24 sec
20	4 sec	2.9 hrs
30	9 sec	124 days
40	16 sec	348 years

Note: Assume 100 operations per second

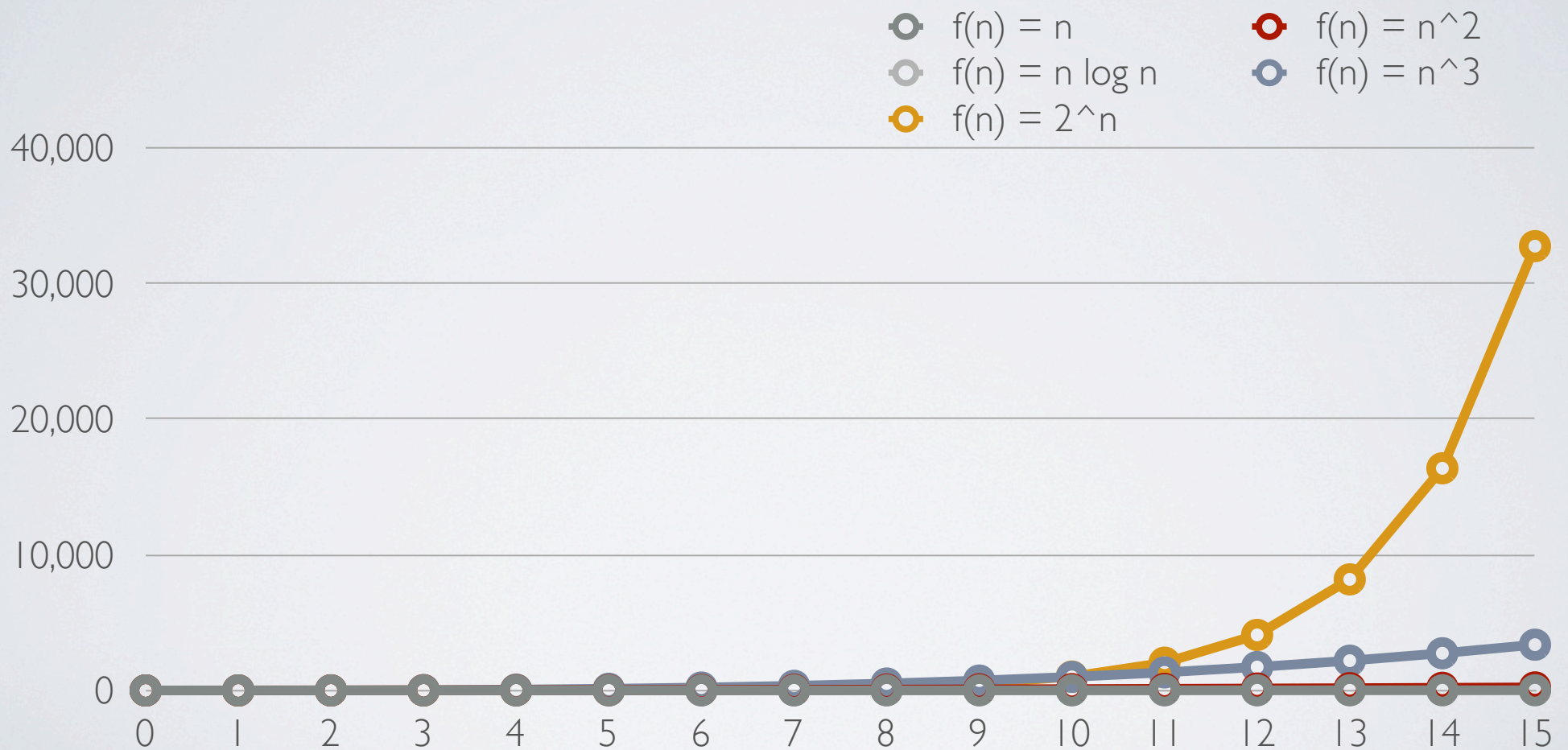
POLYNOMIAL GROWTH RATES



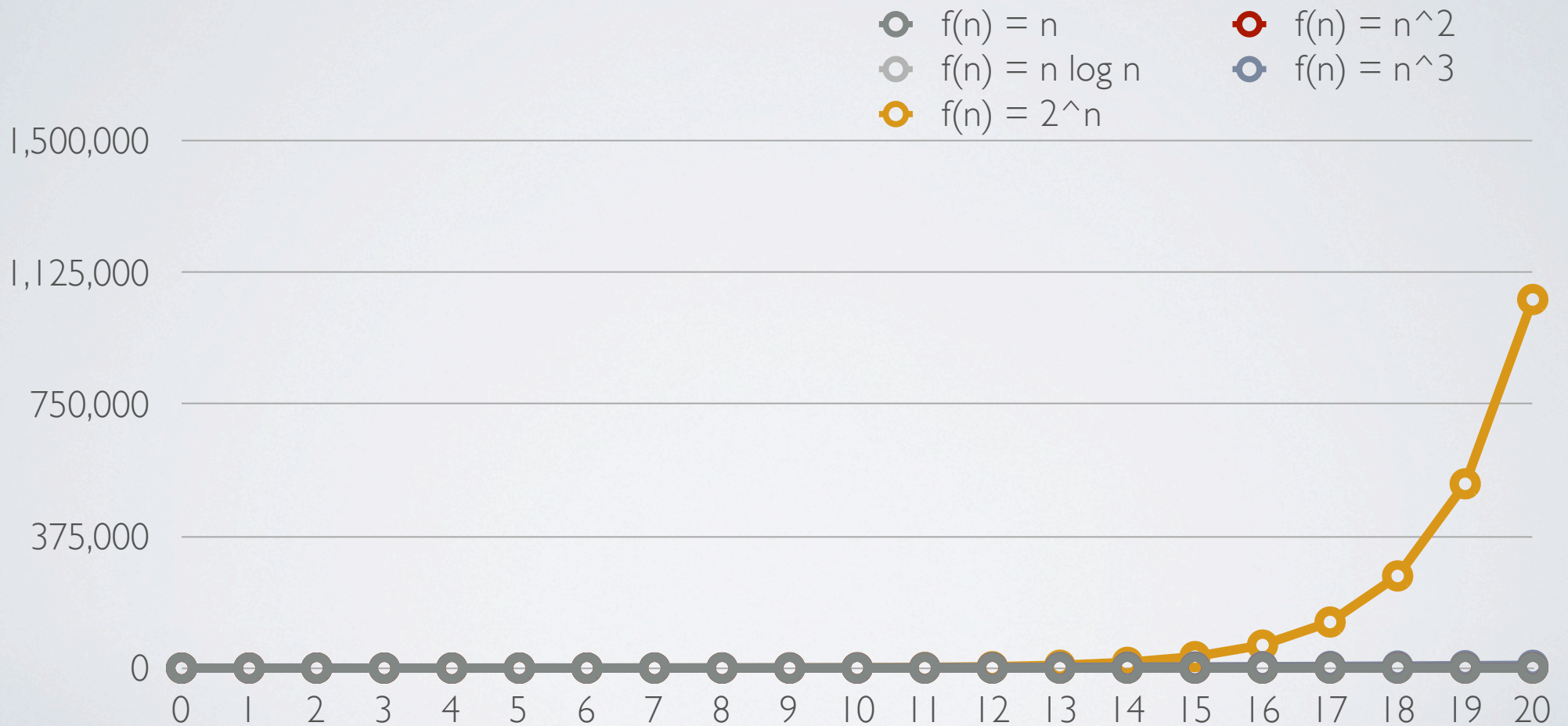
GROWTH RATES



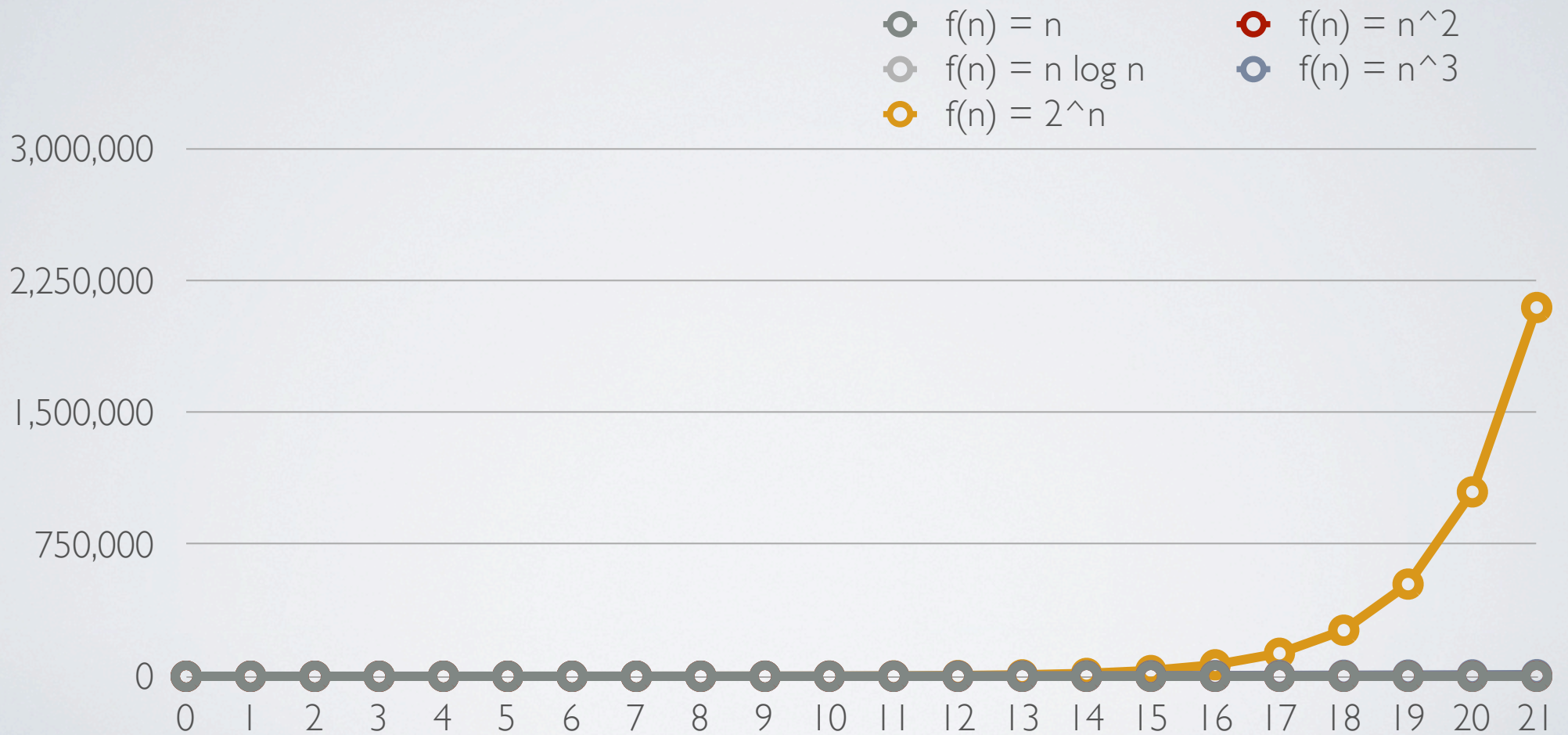
GROWTH RATES



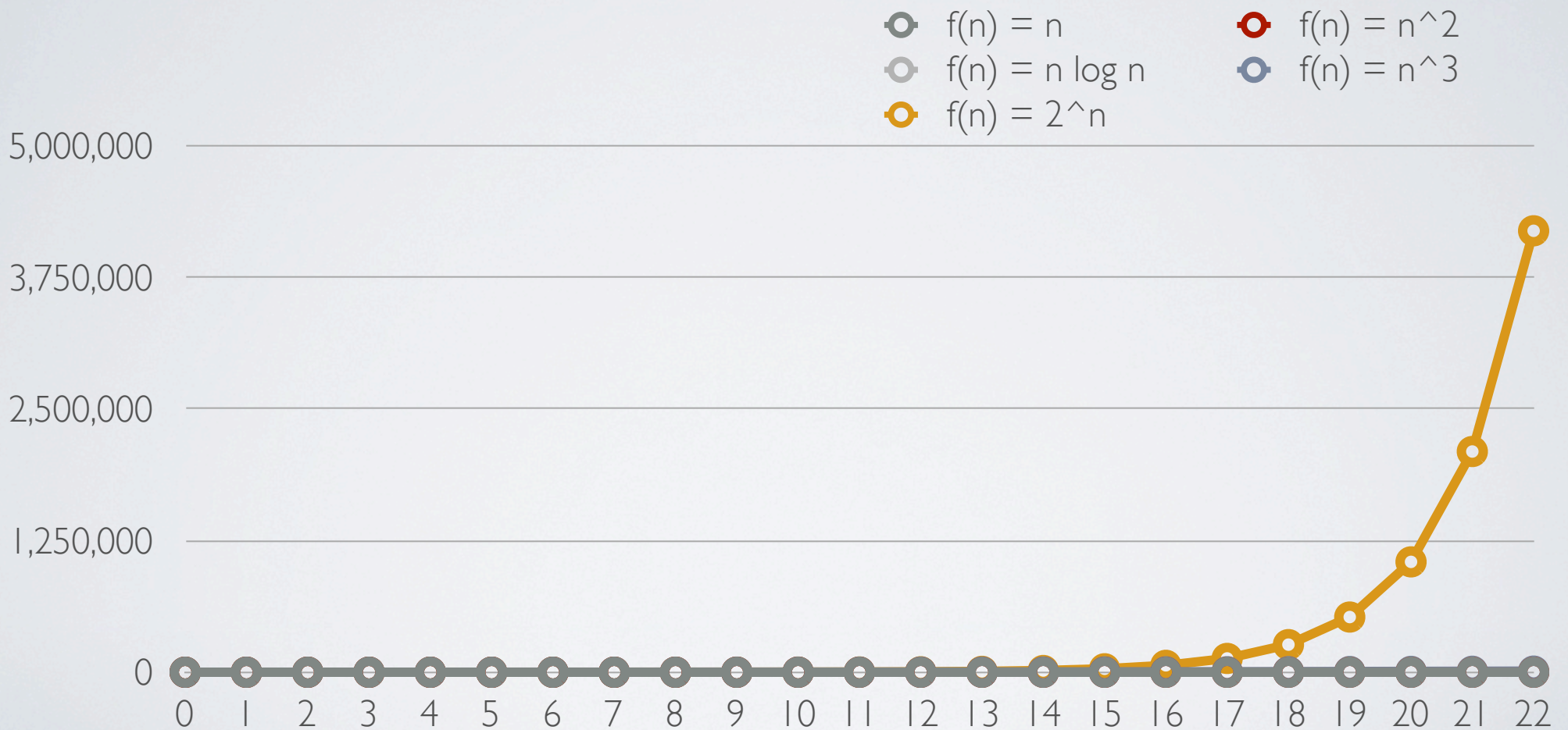
GROWTH RATES



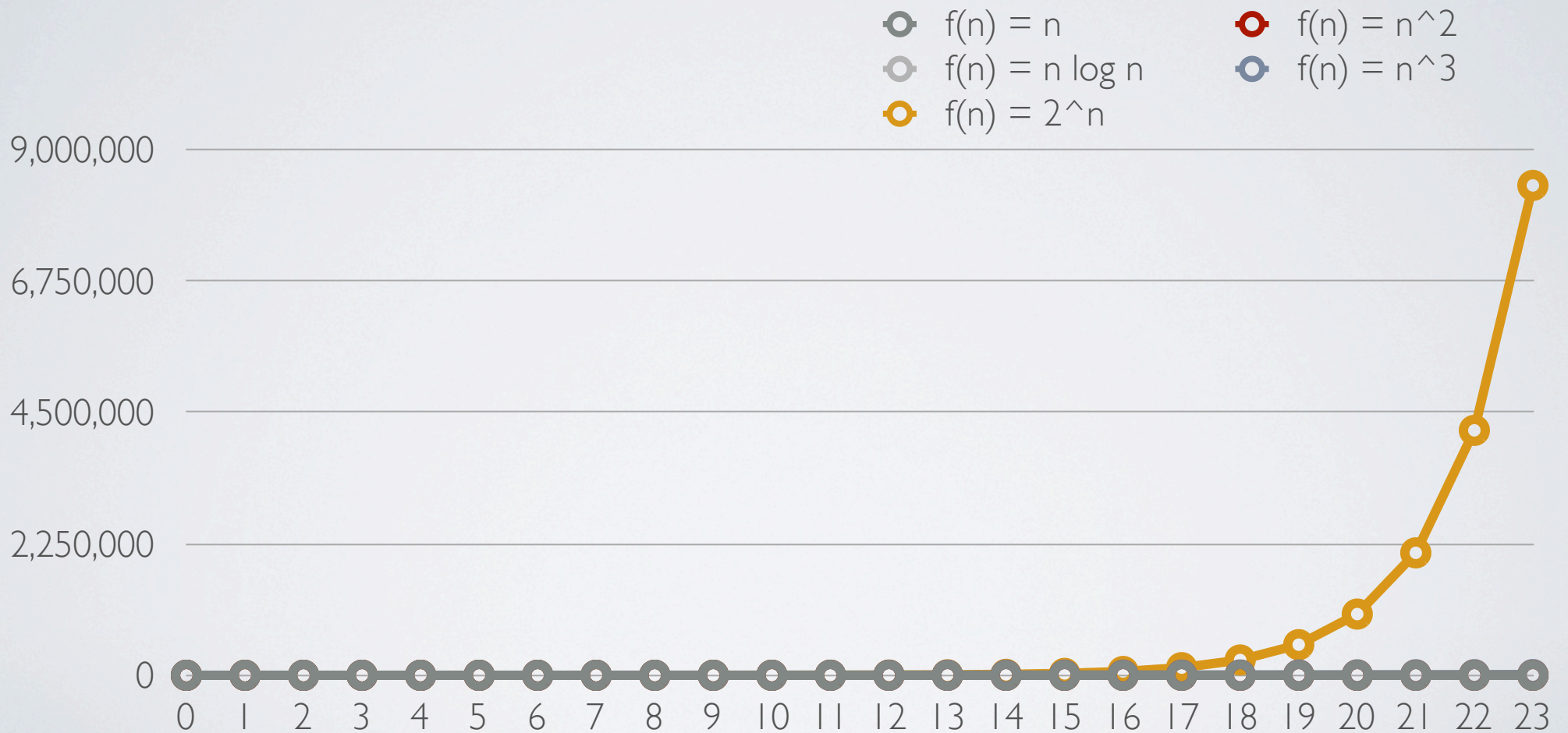
GROWTH RATES



GROWTH RATES



GROWTH RATES



GROWTH RATES



GROWTH RATES



SCHEDULING

Nobody knows if this problem
can be solved in polynomial time

- For a given graph, the problem is to determine a maximum number of vertices that, if all other vertices are removed, are conflict-free.

Optimization Problem

SCHEDULING AS OPTIMIZATION PROBLEM

- Instead of
 - asking for a maximum number of conflict-free vertices to be left we can
 - ask for **a minimum number of vertices to be removed** such that the vertices left are conflict-free

Optimization Problem

THE PROBLEM AS DECISION PROBLEM

Nobody knows if this problem
can be solved in polynomial time either!

- Can we remove up to k vertices such that the vertices left are conflict free?
- An answer to an input for a decision problem (here a graph and a number) is either **yes** or **no**.

THE DECISION PROBLEM

- **What can we do?** Can we remove up to 3 vertices such that the vertices left are conflict free?
 - We can *verify or test* a candidate solution in polynomial time
 - A suggested solution consists of specific vertices that are to be removed.
 - We can, in polynomial time, count and remove these vertices and then test whether all the conflicts are indeed all removed

THE COMPLEXITY CLASS NP

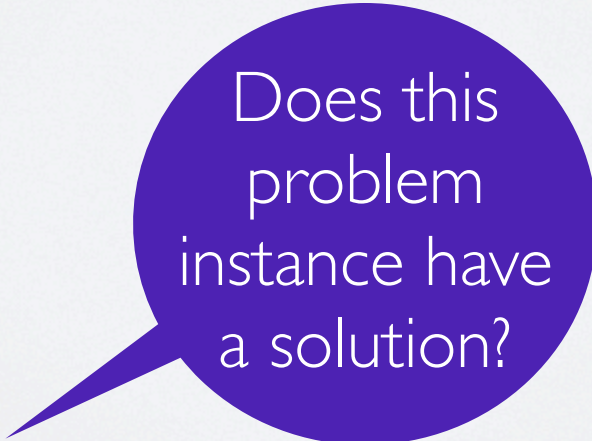
- NP: The class of all (decision) problems for which candidate solutions can be verified in polynomial time
- NP stands for **Nondeterministic Polynomial** time
- The scheduling problem is a member of the class NP

THE COMPLEXITY CLASS NP

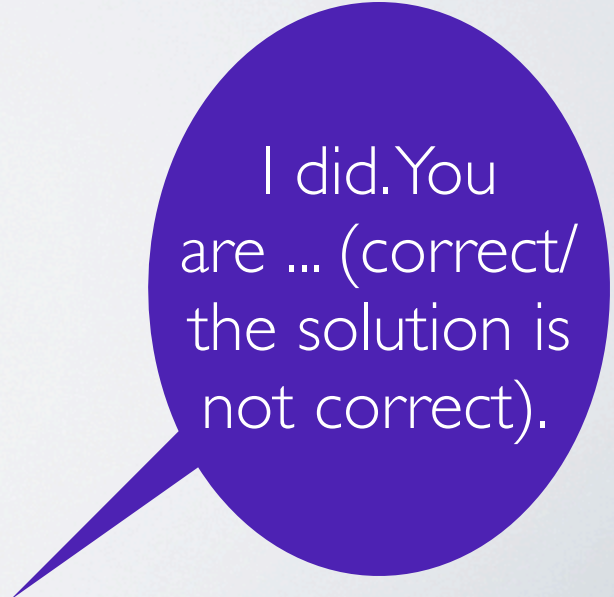
- NP: The class of all (decision) problems for which candidate solutions can be verified in polynomial time



Yes!!
Here is a
solution. Check
yourself!



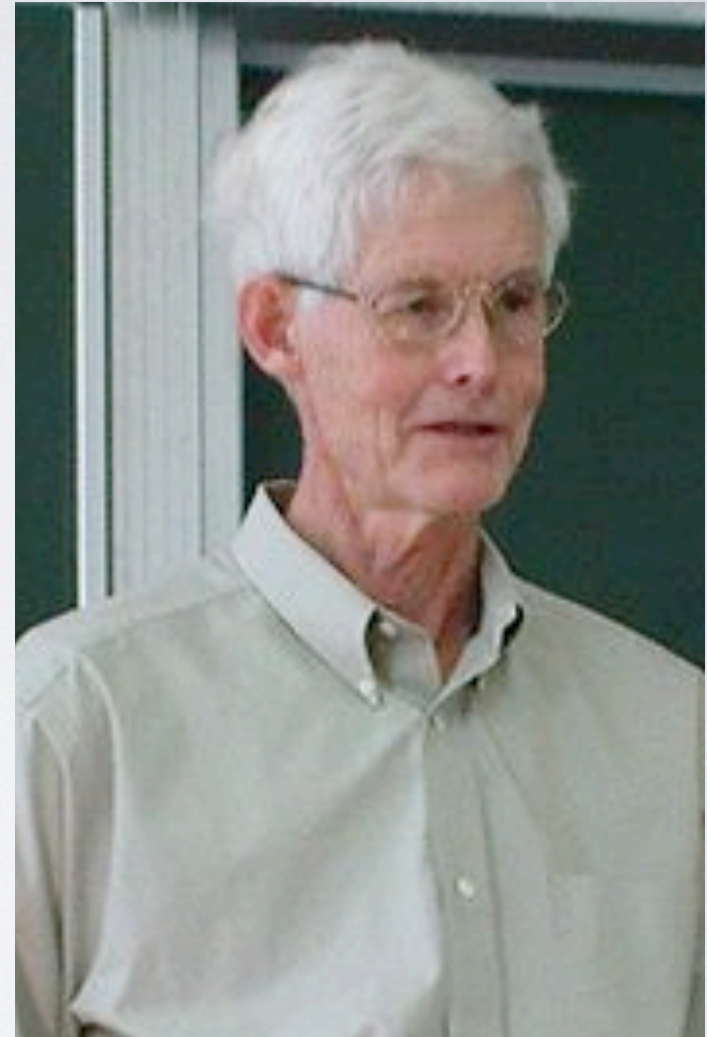
Does this
problem
instance have
a solution?



I did. You
are ... (correct/
the solution is
not correct).

NP-COMPLETE PROBLEMS

- The discussed scheduling problem has a famous property, namely to be *NP-complete*
- There exists many NP-complete problems
- For none of the NP-complete problems it is known whether there exists a polynomial time algorithm that solves it (no matter the input)
- The property NP-complete was introduced by *Stephen Cook*



A \$1,000,000 QUESTION: IS $P = NP$?

- Note that P is a subset of NP , as
 - for every problem that can be solved in polynomial time, also a candidate solution can be verified in polynomial time
- However, most researchers **conjecture** that **P and NP** are **not equal**.
- $P = NP$? is one of the most famous open problems in computer science

IS $P=NP$? A MILLION DOLLAR QUESTION

- The discussed scheduling problem above is also called *Vertex Cover* or *Node Cover*, and was one of the very first problems to be shown *NP-complete*
- **NP-complete problems are special: If just one of them (and there are many!!) can be solved in polynomial time then all of them can be solved in polynomial time.**
- Thus: If just one of them is in P , then $P = NP$ (worth a million dollars!)

MOST MATHEMATICIANS AND COMPUTER SCIENTISTS ...

- conjecture that P is **not** equals to NP
- One of the millenium problems of the Clay Mathematics Institure (http://www.claymath.org/millennium/P_vs_NP/)
- Prize for answering the question: US \$1,000,000

FAMOUS NP-COMPLETE PROBLEMS

- Graph Coloring
- Traveling Salesman Problem
- Dominating Set
- Independent Set
- Clique
- Hamiltonian Circuit
- Eulerian Circuit
- Satisfiability

There exists a very large catalogue of NP-complete problems, most of them have important applications in other areas.

APPLICATION AREAS OF NP-COMPLETE PROBLEMS

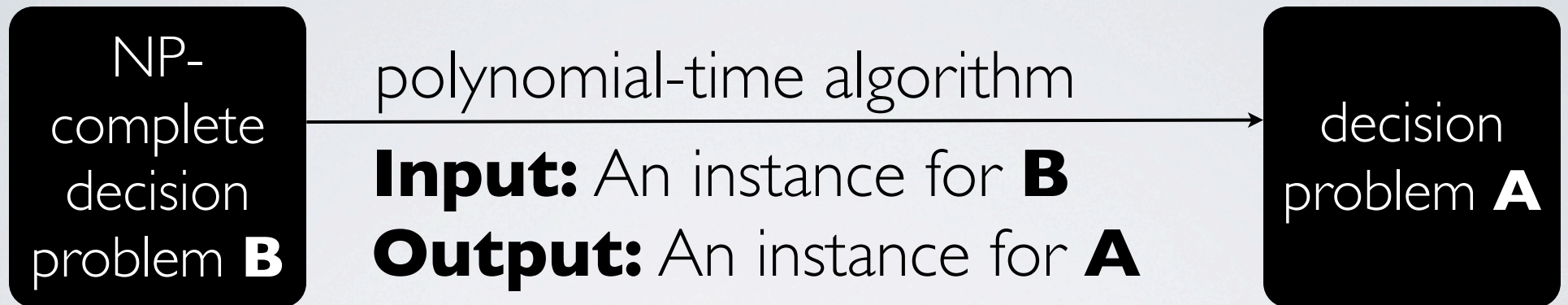
- Conflict resolution (eg. Vertex Cover, Independent Set)
- Scheduling problems (eg. Vertex Cover, Graph Coloring, Traveling Salesman Problem)
- Multiple (Biological) Sequence Alignments
- Determining Evolutionary Trees (eg. Large Parsimony Problem)
- Geomatics (eg. graph coloring)
- and many more

NP-complete problems are everywhere!

NP-COMPLETE PROBLEMS

- Candidate solutions of the decision versions are all verifiable in polynomial time (membership in NP).
- They can be *reduced* to each other:
 - to show that a problem **A** (such as Vertex Cover or Graph Coloring) is NP-complete it is sufficient to transform a known NP-complete problem **B** via a polynomial-time algorithm into **A** such that: a solution of the decision problem for **B** also answers the decision problem for **A** and vice versa.

REDUCTIONS: RELATIONSHIP BETWEEN NP-COMPLETE PROBLEMS



The algorithm must be answer-preserving: yes stays yes, no stays no.

If **A** was solvable in polynomial time, then so was **B**! Why? We input an instance for **B**, transform it into an instance for **A**, solve the instance for **A** in polynomial time, and, since our reduction is answer preserving, we know the answer for our instance for **B**!



ANOTHER COMPUTATIONAL GRAPH PROBLEM

Graph Coloring (Optimization Version)

Input: A graph $G = (V, E)$ consisting of vertices V and edges E

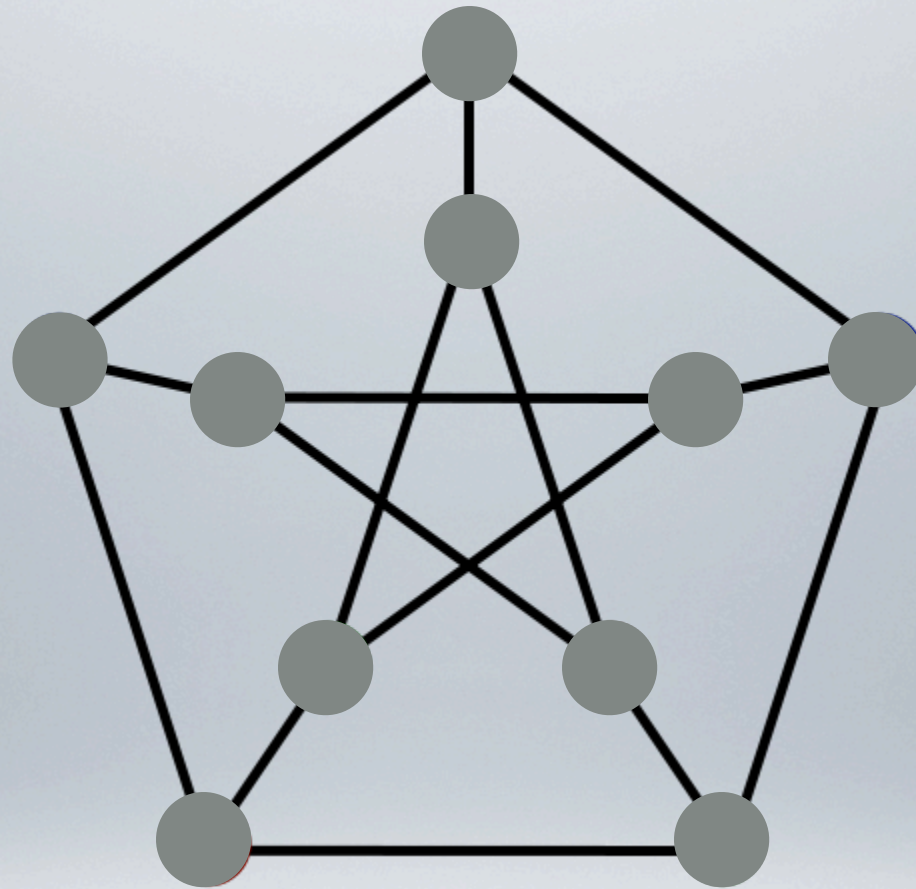
Task: Color the vertices of the graph using at few colors as possible such that no edge connects two vertices of the same color

ANOTHER COMPUTATIONAL GRAPH PROBLEM

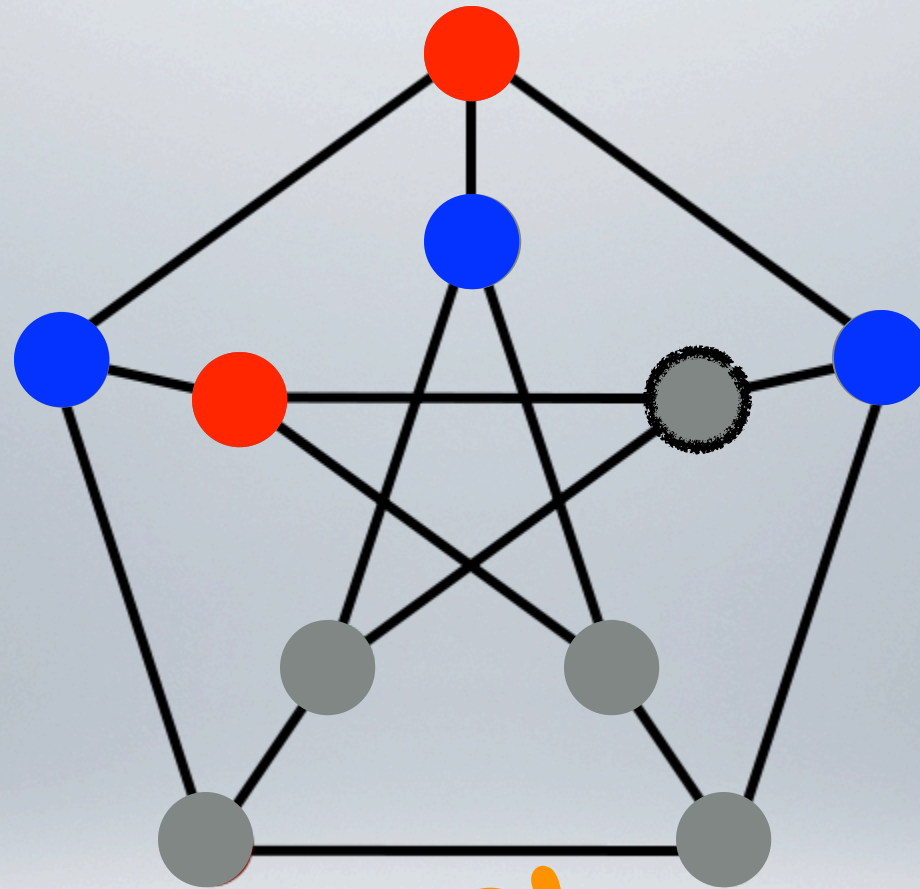
Graph Coloring (Decision Version)

Input: A graph $G = (V, E)$ consisting of vertices V and edges E , a positive integer k

Question: Can we color the vertices of the graph using at most k different colors such that no edge connects two vertices of the same color?

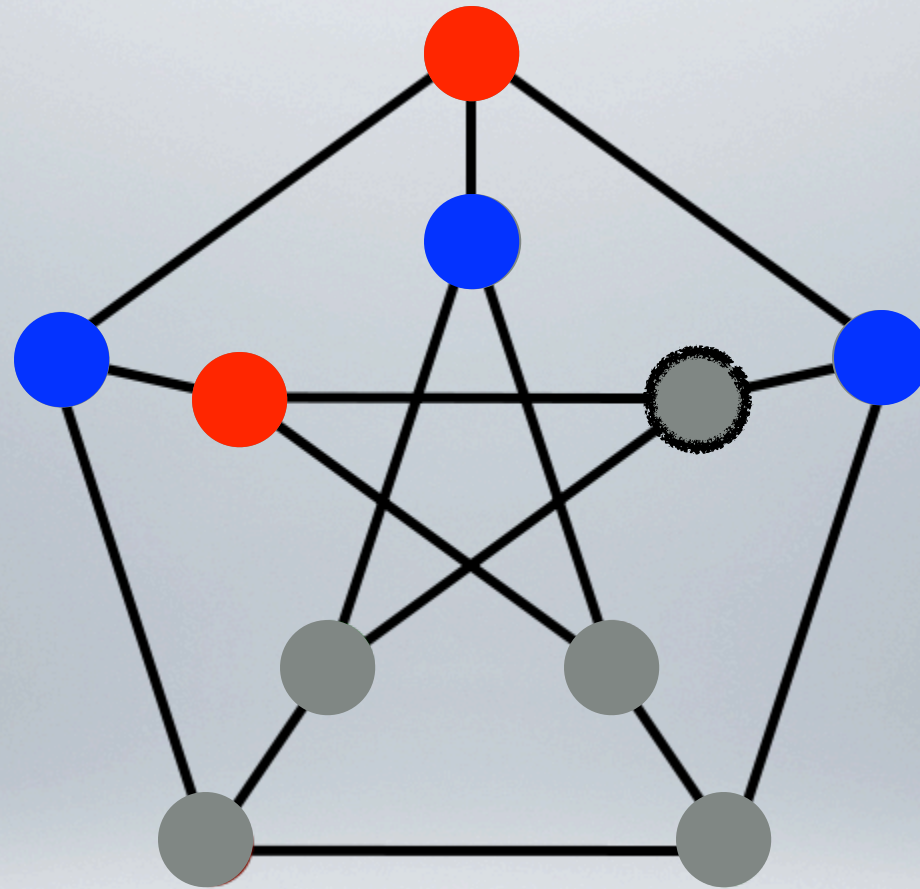


CAN WE 2-COLOR THIS GRAPH?

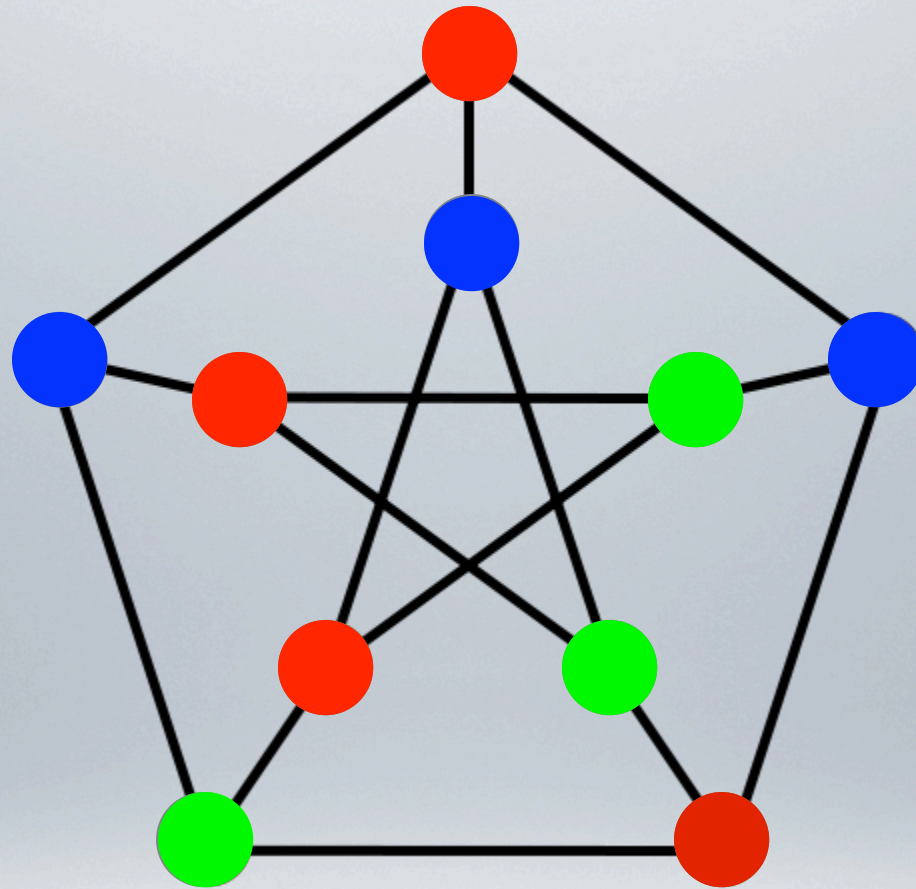


NO!

CAN WE 2-COLOR THIS GRAPH?



CAN WE 3-COLOR THIS GRAPH?



CAN WE 3-COLOR THIS GRAPH?

GRAPH COLORING

- is solvable in polynomial time for $k=2$
- In general, Graph Coloring is NP-complete

THE END