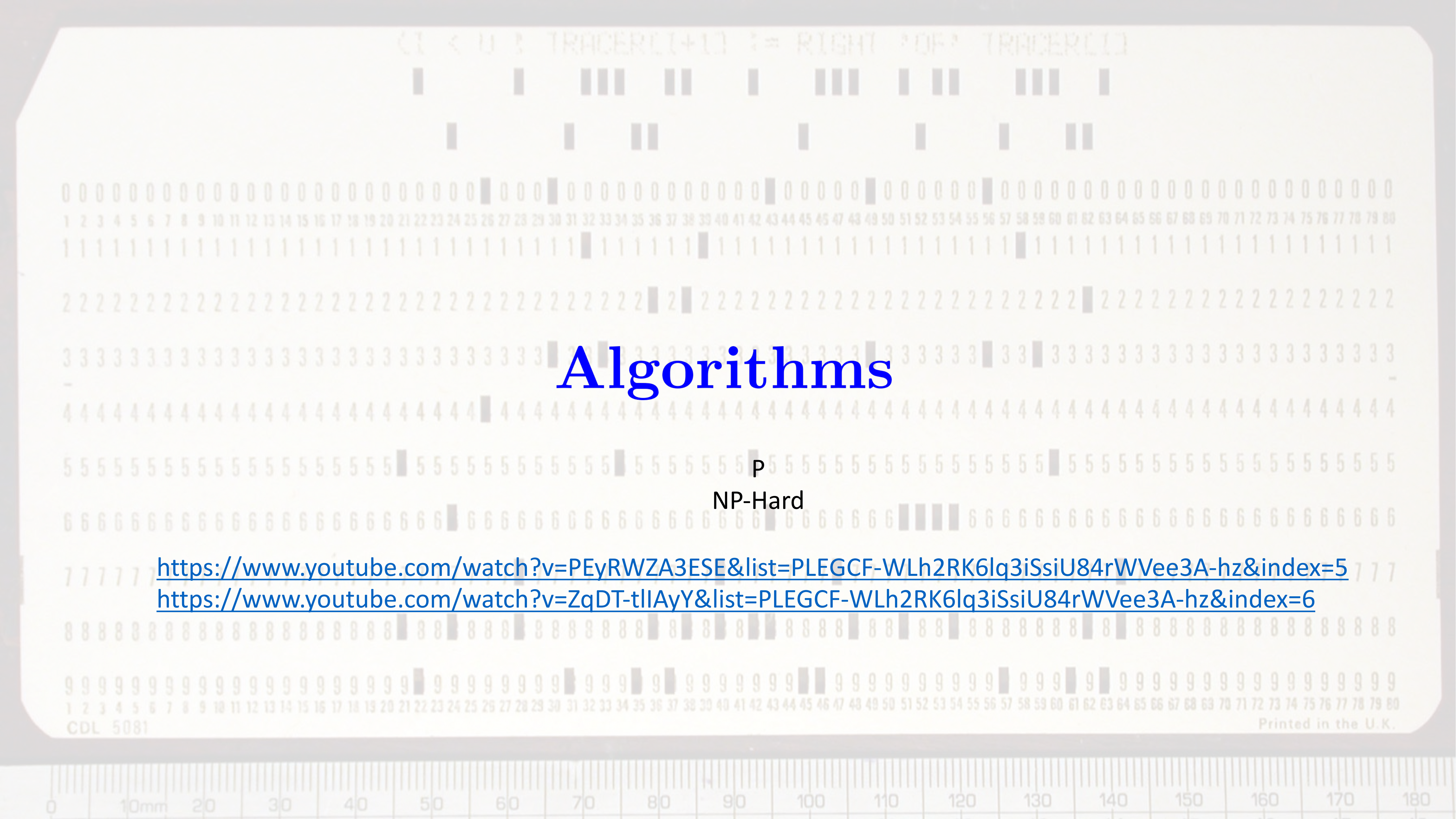


Algorithms

P
NP-Hard

https://www.youtube.com/watch?v=PEyRWZA3ESE&list=PLEGCF-WLh2RK6lq3iSsiU84rWVee3A-hz&index=5
https://www.youtube.com/watch?v=ZqDT-tlIAyY&list=PLEGCF-WLh2RK6lq3iSsiU84rWVee3A-hz&index=6

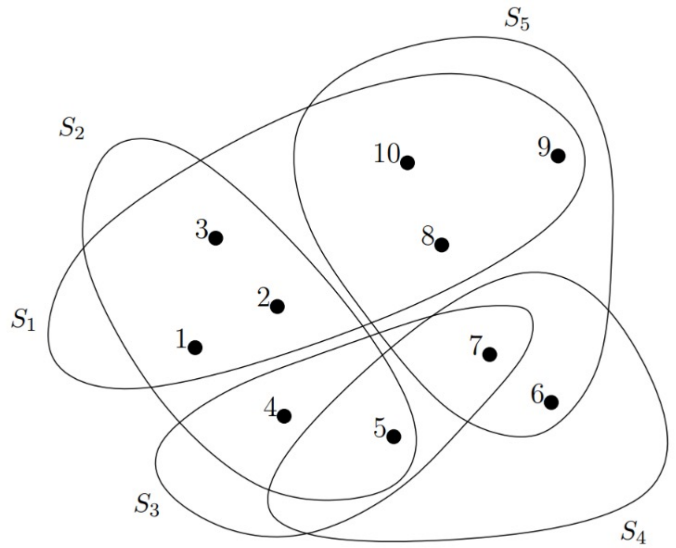
Set Cover

Input. A set X with n elements called the *ground set.*

Subsets: $S_1, S_2, \cdots, S_m \subseteq X$.

Goal. Select the smallest number of sets in $\{S_1, \cdots, S_m\}$ whose union is X.

Example.



$X = \{1, 2, \cdots, 10\}$.

Sets : $S_1, \cdots, S_5$

The smallest collection of these sets that "covers" all elements is $\{S_2, S_5\}$.

Set Cover

Theorem. The greedy algorithm always returns an answer whose size is at most $\ln n$ times that of the optimal solution.

$n = |X|$

Proof. Suppose that the optimal solution has k sets.

Let n_t be the number of uncovered elements in the t^{th} iteration.

Since all elements are uncovered initially $\overset{n_1}{\cancel{n_0}} = n$.

Generalized pigeonhole principle

Since the uncovered elements are covered by the k optimal sets, there is at least one set that covers $\geq n_t/k$ of the these.

$$\text{Thus, } n_{t+1} \leq n_t - \frac{n_t}{k} = n_t\left(1 - \frac{1}{k}\right) < n_t \cdot e^{-1/k}$$

$(\textit{since } 1 - x < e^{-x} \textit{ for } x \neq 0)$

This implies that: $n_t < n \cdot e^{-t/k}$.

For $t = k \ln n$, $n_t < 1$ i.e., $n_t = 0$. $\square$

Under widely believed assumptions, no algorithm has a better guarantee!

Computational Complexity

Question. Which problems can we expect to solve in practice?

OVERSIMPLICATION

Computational Complexity

Question. Which problems can we expect to solve in practice?

Oversimplification: easy to solve vs hard to solve
Easy to solve: can be solved by a polynomial-time algorithm (**P problems**)
Hard to solve: believed to require exponential-time in the worst case (**NP-hard**)

Provide examples of polynomial-time algorithms covered in class

Computational Complexity

which of these problems can be done in polynomial time?

Shortest path
Minimum cut-Max flow
Max-cut
Satisfiability
Prim
Set cover
mergesort

Unfortunately for many important problems we do not know whether there are polynomial time algorithms.

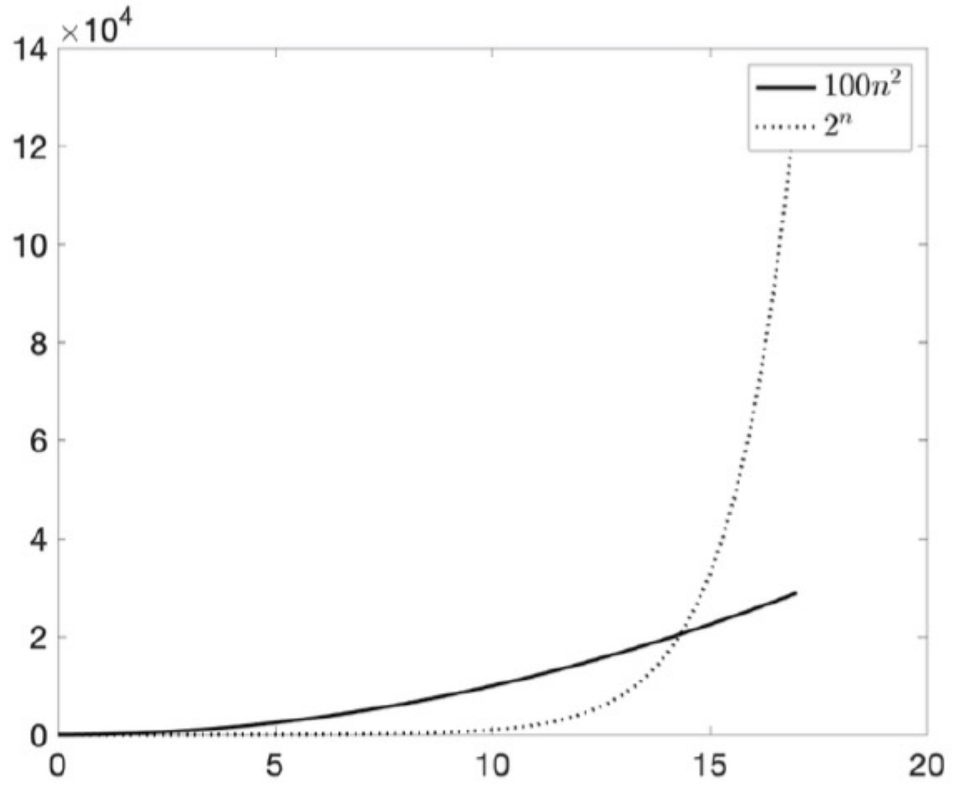
Computational Complexity

Suppose the running time of an algorithm on inputs of size 1,000, 2,000, 3,000, and 4,000 is 5 seconds, 20 seconds, 45 seconds, and 80 seconds, respectively.
Is the order of growth of the running time of the

1. linear,
2. quadratic,
3. cubic, or
4. exponential?

Solution: let n_1=1000
when n_2=2000, time is $20=4*5=(n_2/n_1)^2*5$
n_3=3000, time is $45=9*5= (n_3/n_1)^2*5$
n_4=4000, time is $80=16*5= (n_4/n_1)^2*5$

2. quadratic.
Is this growth problematic?



Computational Complexity

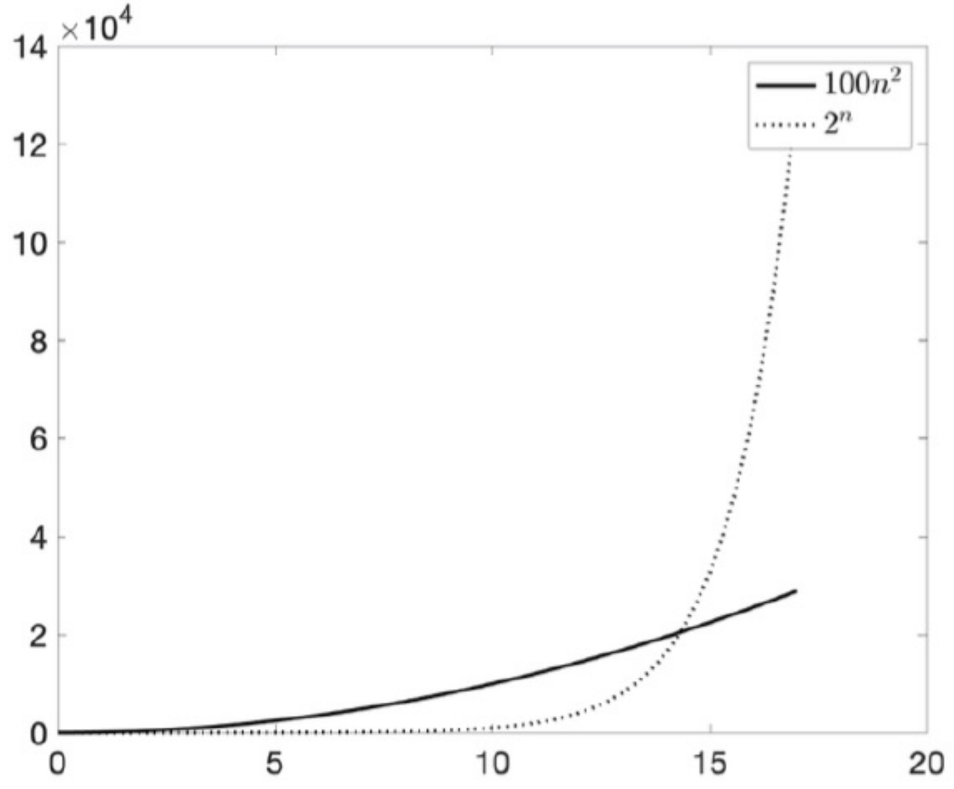
But computers are getting faster?

Let's examine this, what happens if my computational power doubles?

Assume an algorithm takes one hour for a given input n_0.
If the algorithm is linear, what input size **approximately** can we achieve in one hour if the computation power is doubled?
$2\,n_0$

What about an O(n⁴) algorithm?
$(2^{1/4} n_0)^4 = {n_0}^4 + {n_0}^4$
So doubling power → we can do 2 consecutive runs with same input $2 \times (n_0)^4$, **almost** $(2^{1/4} n_0)^4$

Exponential? $(a^n, \text{for } a \geq 2)$
$$a^{n_0} + a^{n_0} = 2a^{n_0} \leq a \times a^{n_0} \leq a^{n_0+1}$$



Computational Complexity

Easy Problems (P)

<u>A Polynomial-time algorithm:</u>
Runs in O(n^d) for some constant d that is independent of n

<u>A problem is polynomial time solvable</u>:
There exists a polynomial time algorithm that solves it

What do we mean then if we say “not a polynomial-time algorithm”?
Not even a O($n^{1,000,000}$)

Computational Complexity

Hard Problems (NP hard)

How to define a hard problem
How to prove that a problem does not have a polynomial-time algorithm?

Mathematically- Best- not available

Circumstancial:
Many brilliant scientists worked on it for a long time – no poly-time alg. yet

Stronger evidence:
A polynomial-time algorithm would automatically solve thousands of problems that have yet resisted all efforts of a lot of scientists

In fact, all these problems variations of the same problem

Computational Complexity

So, what are NP-Hard Problems?

We agree that there is strong evidence on intractability.

Definition: A problem is NP-hard if a polynomial-time algorithm solving it would refute the P≠NP conjecture.

But what is P ≠ NP?
Checking whether a particular input is a solution is easier than coming up with a solution for the problem.

Computational Complexity

P : set of decision problems for which there is a polynomial time algorithm

NP : We say that a decision problem is in NP if for any instance of the problem for which the answer is 'YES', there is *certificate* using which a *verifier* can verify this in polynomial time.

P stands for "polynomial" and NP stands for "nondeterministic polynomial".

Example. COMPOSITES: Is a given number N composite?

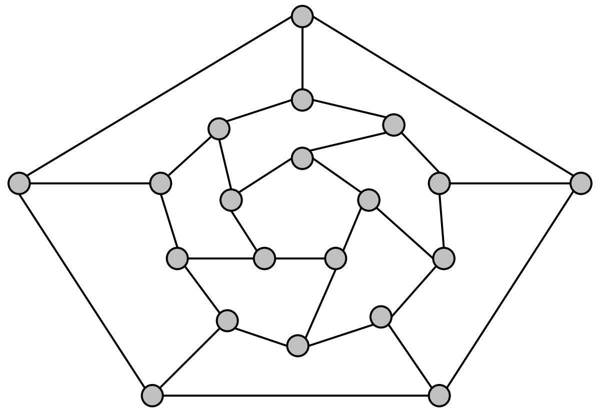
Is this problem in NP? *Yes! Given a factorization we can easily check if it is correct.*

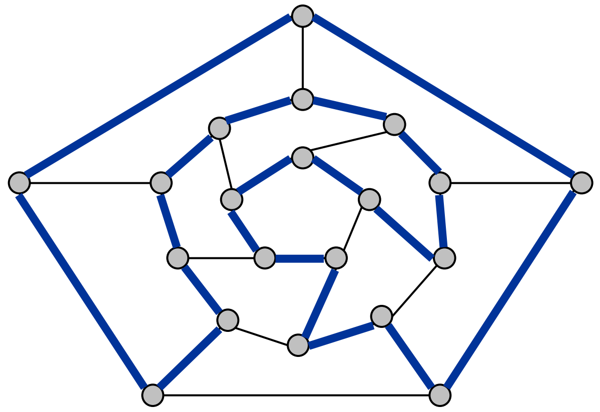
This problem is also known to be in P. *This was a breakthrough result!*

Computational Complexity

HAMILTONIAN CYCLE:

Does a given graph have a simple cylce that visits all nodes?





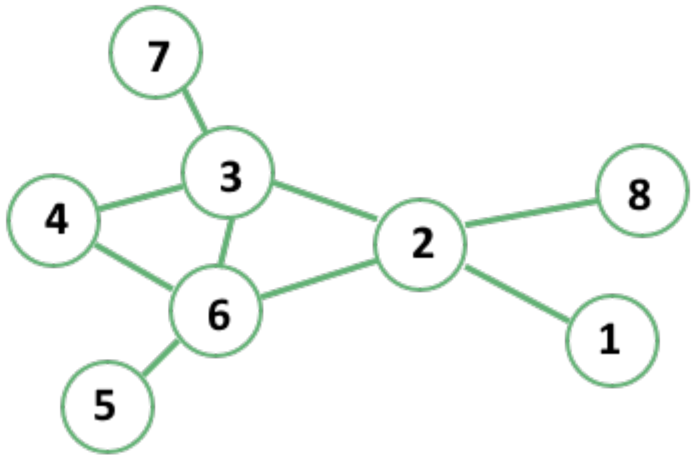
Is this problem in NP? *Yes, given the cycle, we can easily verify it.*

Note that we only need to verify when the answer is ‘YES’.

Independent Set

Let $G = (V, E)$ be an undirected graph.

An ***independent set*** in the graph is a subset $U \subseteq V$ of the vertices s.t. there is no edge in G between any pair of vertices in U.



Consider the graph on the right.

Is $\{1, 3, 7\}$ an independent set? *No* Is $\{1, 3, 5\}$ an independent set? *Yes*

What is the size of the largest independent set? 5 $(\{1, 4, 5, 7, 8\})$

A formulation of the same problem is: Is there an independent set of size>=k? (k∈ $\{n, .., 1\}$)

And an instance of the problem would be, for a given value of k, is there a solution? (O(n^k) complexit)

Computational Complexity

NP-hard are believed to have no polynomial time solutions
So, Given an NP hard problem, do we have strategies to solve them?

General purpose: Can solve a subset of the input only (that is polynomially solvable)

Correctness: Lower the accuracy of the result

Speed: runs in (super) polynomial-time for better correctness

Next…

- Reduction examples
- NP-hard examples