1.1 A First Problem: Stable Matching

Matching Residents to Hospitals

Goal. Given a set of preferences among hospitals and medical students, design a self-reinforcing admissions process.

Unstable pair: applicant x and hospital y are unstable if:
- x prefers y to its assigned hospital.
- y prefers x to one of its admitted students.

Stable assignment. Assignment with no unstable pairs.
- Natural and desirable condition.
- Individual self-interest will prevent any applicant/hospital deal from being made.

Stable Matching Problem

Goal. Given n men and n women, find a "suitable" matching.

- Participants rate members of opposite sex.
- Each man lists women in order of preference from best to worst.
- Each woman lists men in order of preference from best to worst.

favorite ↓ ... least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ ... least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

Stable Matching Problem

Perfect matching: everyone is matched monogamously.

- Each man gets exactly one woman.
- Each woman gets exactly one man.

Stability: no incentive for some pair of participants to undermine assignment by joint action.

- In matching M, an unmatched pair m-w is unstable if man m and woman w prefer each other to their current partner.
- Unstable pair m-w could each improve by eloping.

Stable matching: perfect matching with no unstable pairs.

Stable matching problem. Given the preference lists of n men and n women, find a stable matching if one exists.

Stable Roommate Problem

Q. Do stable matchings always exist?
A. Not obvious a priori.

Stable roommate problem.
- 2n people; each person ranks others from 1 to 2n-1.
- Assign roommate pairs so that no unstable pairs.

	1st	2nd	3rd
Adam	B	C	D
Bob	C	A	D
Chris	A	B	D
Doofus	A	B	C

A-B, C-D $\Rightarrow$ B-C unstable
A-C, B-D $\Rightarrow$ A-B unstable
A-D, B-C $\Rightarrow$ A-C unstable

Observation. Stable matchings do not always exist for stable roommate problem.

Propose-And-Reject Algorithm

Propose-and-reject algorithm. [Gale-Shapley 1962] Intuitive method that guarantees to find a stable matching.

```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

Proof of Correctness: Termination

Observation 1. Men propose to women in decreasing order of preference.

Observation 2. Once a woman is matched, she never becomes unmatched; she only "trades up."

Claim. Algorithm terminates after at most n^2 iterations of while loop.
Pf. Each time through the while loop a man proposes to a new woman. There are only n^2 possible proposals. ▪

	1st	2nd	3rd	4th	5th
Victor	A	B	C	D	E
Wyatt	B	C	D	A	E
Xavier	C	D	A	B	E
Yancey	D	A	B	C	E
Zeus	A	B	C	D	E

	1st	2nd	3rd	4th	5th
Amy	W	X	Y	Z	V
Bertha	X	Y	Z	V	W
Clare	Y	Z	V	W	X
Diane	Z	V	W	X	Y
Erika	V	W	X	Y	Z

$n(n-1) + 1$ proposals required

Proof of Correctness: Perfection

Claim. All men and women get matched.

Pf. (by contradiction)

- Suppose, for sake of contradiction, that Zeus is not matched upon termination of algorithm.
- Then some woman, say Amy, is not matched upon termination.
- By Observation 2, Amy was never proposed to.
- But, Zeus proposes to everyone, since he ends up unmatched. ▪

Proof of Correctness: Stability

Claim. No unstable pairs.

Pf. (by contradiction)

- Suppose A-Z is an unstable pair: each prefers each other to partner in Gale-Shapley matching S*.

- Case 1: Z never proposed to A.
  - ⇒ Z prefers his GS partner to A.
  - ⇒ A-Z is stable.

men propose in decreasing order of preference

S*
Amy-Yancey
Bertha-Zeus
. . .

- Case 2: Z proposed to A.
  - ⇒ A rejected Z (right away or later)
  - ⇒ A prefers her GS partner to Z. ← women only trade up
  - ⇒ A-Z is stable.

- In either case A-Z is stable, a contradiction. ▪

Summary

Stable matching problem. Given n men and n women, and their preferences, find a stable matching if one exists.

Gale-Shapley algorithm. Guarantees to find a stable matching for any problem instance.

Q. How to implement GS algorithm efficiently?

Q. If there are multiple stable matchings, which one does GS find?

Efficient Implementation

Efficient implementation. We describe $O(n^2)$ time implementation.

Representing men and women.

- Assume men are named 1, …, n.
- Assume women are named 1', …, n'.

Engagements.

- Maintain a list of free men, e.g., in a queue.
- Maintain two arrays `wife[m]`, and `husband[w]`.
  - set entry to `0` if unmatched
  - if m matched to w then `wife[m]=w` and `husband[w]=m`

Men proposing.

- For each man, maintain a list of women, ordered by preference.
- Maintain an array `count[m]` that counts the number of proposals made by man `m`.

Efficient Implementation

Women rejecting/accepting.

- Does woman `w` prefer man `m` to man `m'`?
- For each woman, create inverse of preference list of men.
- Constant time access for each query after O(n) preprocessing.

Amy	1st	2nd	3rd	4th	5th	6th	7th	8th
Pref	8	3	7	1	4	5	6	2

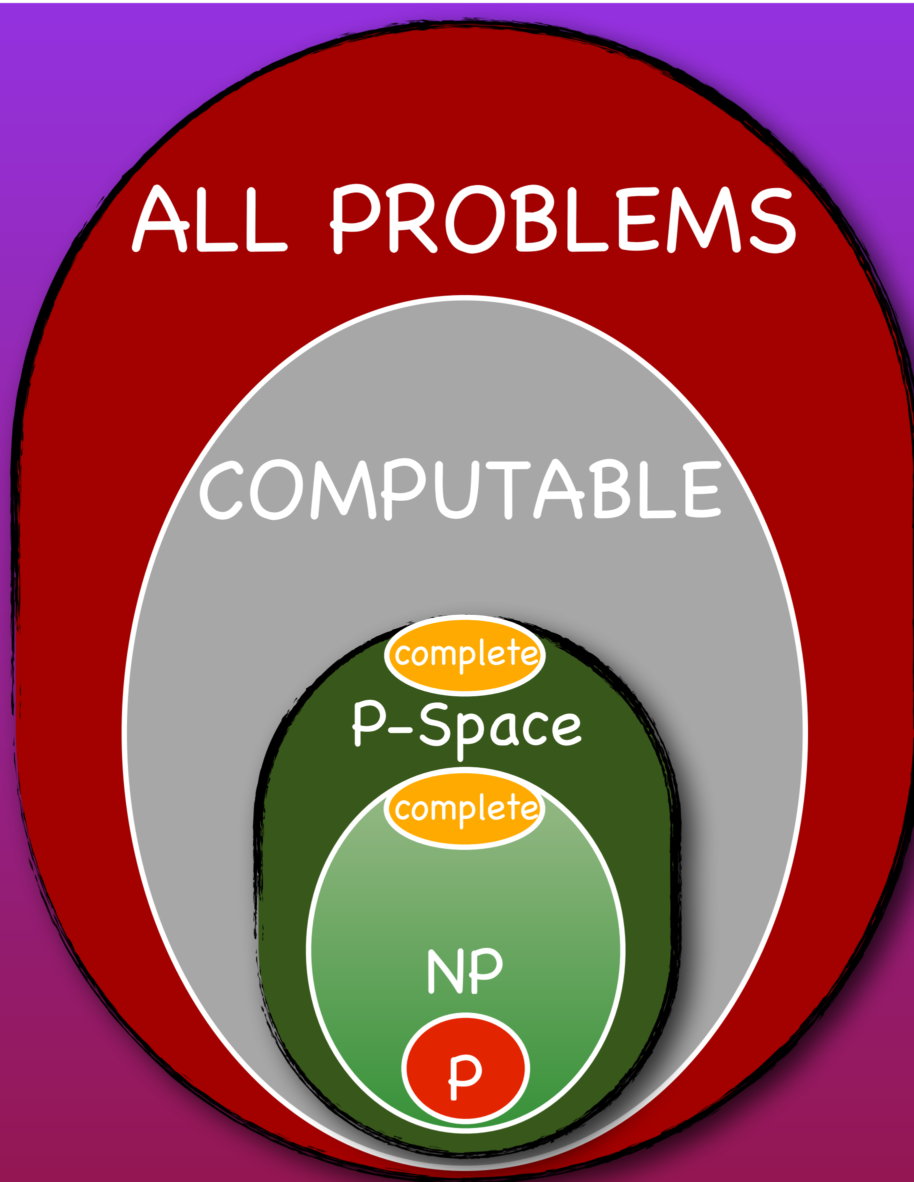
Amy	1	2	3	4	5	6	7	8
Inverse	4th	8th	2nd	5th	6th	7th	3rd	1st

```
for i = 1 to n
   inverse[pref[i]] = i
```

Amy prefers man 3 to 6
since `inverse[3]` < `inverse[6]`

2 7

A few Computability Classes



Polynomial-Time

Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes 2^N time or worse for inputs of size N.
- Unacceptable in practice.

N! for stable matching with N men and N women

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor C.

> There exists constants a > 0 and d > 0 such that on every input of size N, its running time is bounded by $a \cdot N^d$ steps.

Def. An algorithm is poly-time if the above scaling property holds.

choose $C = 2^d$

Property: poly-time is invariant over *all* computer models.

Average/Worst-Case Analysis

Worst case running time. Obtain bound on largest possible running time of algorithm on any input of a given size N.

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.
- For probabilistic algorithms, we take the worst average running time.

Average case running time. Obtain bound on running time of algorithm on random input as a function of input size N.

- Hard (or impossible) to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other input distributions.

Worst-Case Polynomial-Time

Def. An algorithm is efficient if its running time is polynomial.

Justification: It really works in practice!

- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

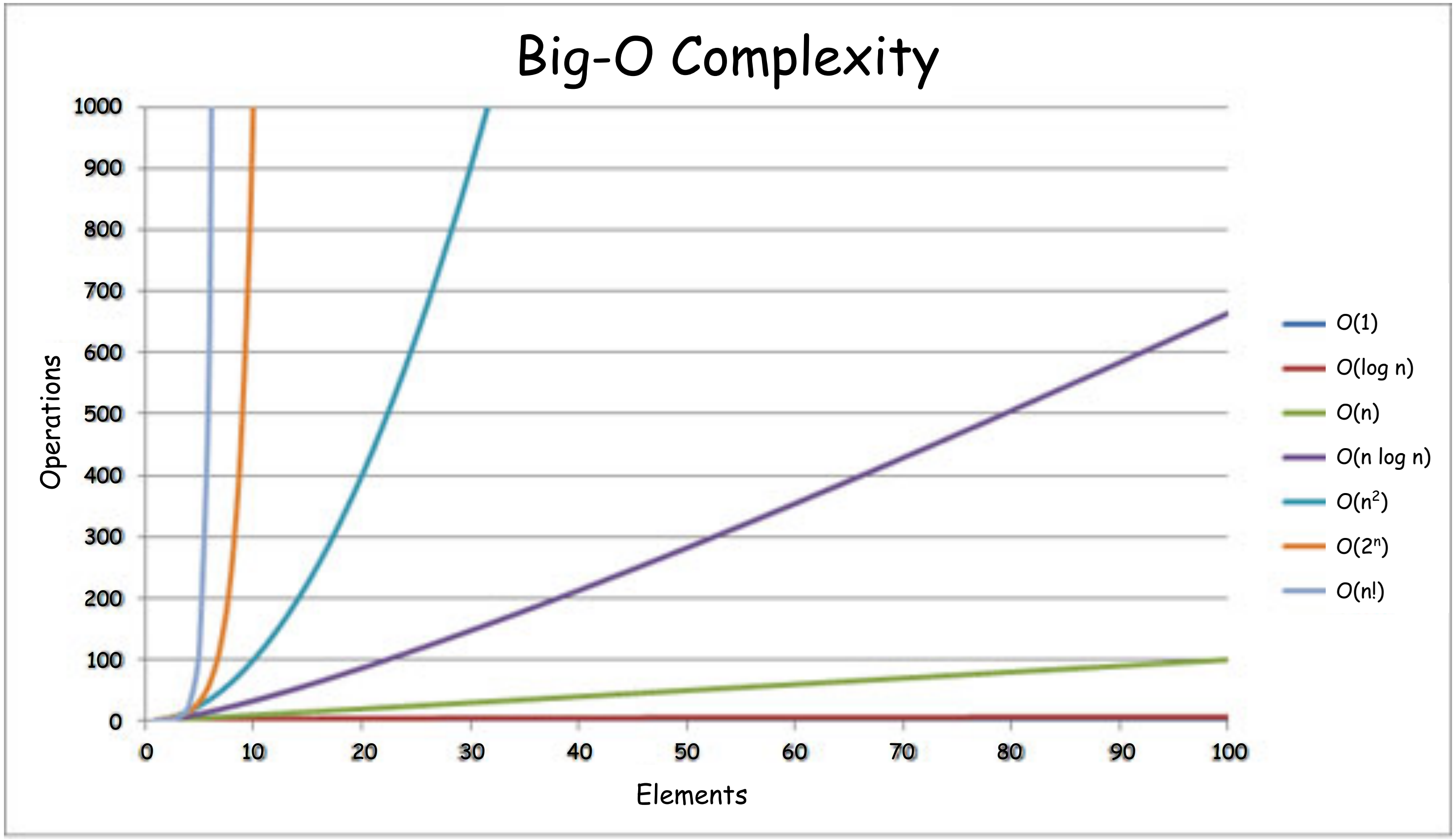
Exceptions.

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

Primality testing

simplex method
Unix grep

Why It Matters



2.2 Asymptotic Order of Growth

Asymptotic Order of Growth

Let $f:\mathbb{N}\to\mathbb{R}^+$ be a function, we define

Upper bounds.

$$O(f) = \{\ g:\mathbb{N}\to\mathbb{R}^+ \mid \exists c\in\mathbb{R}^+, n_0\in\mathbb{N} \text{ s.t. } \forall n \geq n_0\ [\ g(n) \leq c \cdot f(n)\ ]\ \}.$$

Lower bounds.

$$\Omega(f) = \{\ g:\mathbb{N}\to\mathbb{R}^+ \mid \exists c\in\mathbb{R}^+, n_0\in\mathbb{N} \text{ s.t. } \forall n \geq n_0\ [\ g(n) \geq c \cdot f(n)\ ]\ \}.$$

Tight bounds.

$$\Theta(f) = O(f) \cap \Omega(f).$$

Ex: $T(n) = 32n^2 + 17n + 32.$

$$T(n) \in O(n^2), O(n^3), \Omega(n^2), \Omega(n), \text{ and } \Theta(n^2)\ .$$
$$T(n) \notin O(n), \Omega(n^3), \Theta(n), \text{ or } \Theta(n^3).$$

Notation

Abuse of notation. $T(n) = O(f(n))$.

- Not transitive:
  - $f(n) = 5n^3$; $g(n) = 3n^2$
  - $f(n) = O(n^3)$ and $g(n) = O(n^3)$ but $f(n) \neq g(n)$.
- Better notation: $T(n) \in O(f(n))$.
- Acceptable notation: $T(n)$ **is** $O(f(n))$. (if scared by $\in$!)

Meaningless statement. Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.

- Statement doesn't "type-check".
- Precisely, $f(n)=1 \in O(n \log n)$, therefore "at least one comparison".
- Use Ω for lower bounds: "at least $\Omega(n \log n)$ comparisons".
- "requires at least $cn \log n$ comparisons for $c>0$ and all large enough n".

Limit theorems.

Let $f, g: \mathbb{N} \to \mathbb{R}^+$ be functions, such that

$$\lim_{n \to \infty} f(n)/g(n) = c \in \mathbb{R}^+,$$

then $f \in \Theta(g)$, $g \in \Theta(f)$, $\Theta(f) = \Theta(g)$

$$\lim_{n \to \infty} f(n)/g(n) = 0,$$

then $f \in O(g)$, $f \notin \Omega(g)$,
$O(f) \subsetneq O(g)$, $\Omega(g) \subsetneq \Omega(f)$

Properties

Let $f,g:\mathbb{N}\to\mathbb{R}^+$ be functions

Transitivity.

- If $f \in O(g)$ and $g \in O(h)$ then $f \in O(h)$ since $O(f)\subset O(g)\subset O(h)$.
- If $f \in \Omega(g)$ and $g \in \Omega(h)$ then $f \in \Omega(h)$ since $\Omega(f)\subset\Omega(g)\subset\Omega(h)$.
- If $f \in \Theta(g)$ and $g \in \Theta(h)$ then $f \in \Theta(h)$ since $\Theta(f)\subset\Theta(g)\subset\Theta(h)$.

Additivity.

- If $f \in O(h)$ and $g \in O(h)$ then $f + g \in O(h)$
  since $f(n) < c_f\, h(n)$, $g(n) < c_g\, h(n) \Rightarrow f(n) + g(n) < (c_f+c_g)\, h(n)$.
- If $f \in \Omega(h)$ and $g \in \Omega(h)$ then $f + g \in \Omega(h)$.
- If $f \in \Theta(h)$ and $g \in O(h)$ then $f + g \in \Theta(h)$.

Consequence:

- $f + g \in O(\max\{f,g\})$ since $f + g \leq 2\max\{f,g\}$.
- $f + g \in \Omega(\max\{f,g\})$ since $f + g \geq \max\{f,g\}$.
- $f + g \in \Theta(\max\{f,g\})$ since $\max\{f,g\} \leq f + g \leq 2\max\{f,g\}$.

Asymptotic Bounds for Some Common Functions

Polynomials. $a_0 + a_1 n + \ldots + a_d n^d \in \Theta(n^d)$ if $a_d > 0$.

Polynomial time. Running time $\in O(n^d)$ for some constant d independent of the input size n.

Logarithms. $O(\log_a n) \in O(\log_b n)$ for any constants a, b > 0.

can avoid specifying the base

Logarithms. For every x > 0, $\log n \in O(n^x)$.

log grows slower than every polynomial

Exponentials. For every r > 1 and every d > 0, $n^d \in O(r^n)$.

every exponential grows faster than every polynomial

2.4 A Survey of Common Running Times

Linear Time: O(n)

Linear time. Running time is proportional to input size.

Computing the maximum. Compute minimum of n numbers $a_1, \ldots, a_n$.

```
min ← a1
for i = 2 to n {
   if (ai < min)
      min ← ai
}
```

O(n log n) Time

O(n log n) time. Arises in divide-and-conquer algorithms.

also referred to as linearithmic time

Sorting. **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

Closest Points on a line. Given n numbers $x_1, \ldots, x_n$, what is the smallest distance x_i-x_j between any two points?

O(n log n) solution. Sort the n numbers. Scan the sorted list in order, identifying the minimum gap between successive points.

Quadratic Time: $O(n^2)$

Quadratic time. Enumerate all pairs of elements.

Closest pair of points. Given a list of n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest.

$O(n^2)$ solution. Try all pairs of points.

```
min ← (x1 - x2)² + (y1 - y2)²
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - xj)² + (yi - yj)²
      if (d < min)
         min ← d
   }
}
```

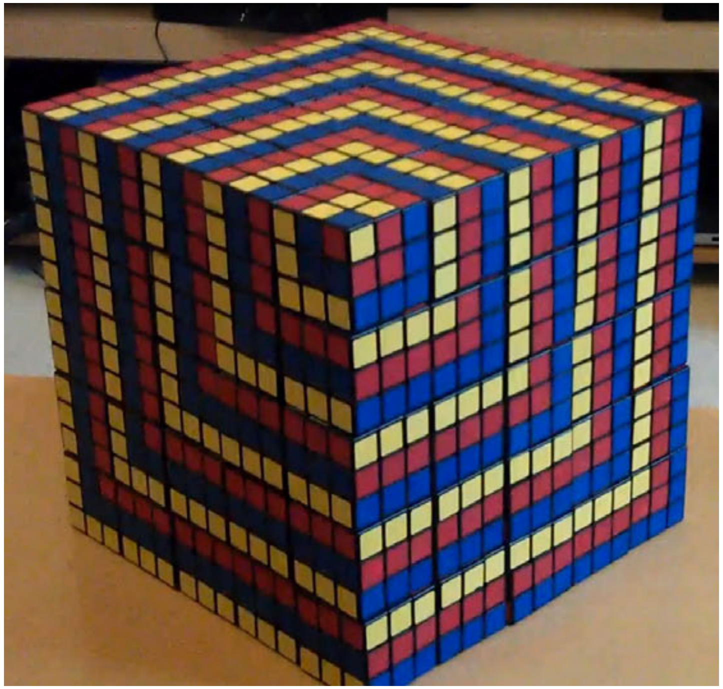
don't need to take square roots

Remark. This algorithm is $\Omega(n^2)$ and it seems inevitable in general, but this is just an illusion: : $\Theta(n \log n)$ is actually possible and optimal... see chapter 5

Quadratic Time: $O(n^2)$

Quadratic time. Solve $O(n^2)$ independent sub-puzzles each in constant-time.

nxnxn Rubik's cube. Given a scrambled nxnxn cube, put it in solved configuration.



Remark. This algorithm is $\Omega(n^2)$ and it seems inevitable in general, but this is just an illusion: $\Theta(n^2/\log n)$ is actually possible and optimal…

Cubic Time: $O(n^3)$

Cubic time. Enumerate all triples of elements.

Matrix multiplication. Given two nxn matrices of numbers A,B, what is their matrix product C ?

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$O(n^3)$ solution. For each entry c_{ij} compute as below.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Polynomial Time: $O(n^k)$ Time

Independent set of size k. Given a graph, are there k nodes such that no two are joined by an edge?

k is a constant

$O(n^k)$ solution. Enumerate all subsets of k nodes.

```
foreach subset S of k nodes {
   if (S is an independent set)
      report S
   }
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets = $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \le \frac{n^k}{k!}$
- $O(k^2\, n^k / k!) = O(n^k)$.

poly-time for k=17,
but not practical

Exponential Time

Independent set. Given a graph, what is the maximum size of an independent set?

$O(n^2 \, 2^n)$ solution. Enumerate all subsets.

```
S* ← ∅
foreach subset S of nodes {
   if (S is an independent set and |S|>|S*|)
      update S* ← S
   }
}
```