CSE 421: Introduction to Algorithms

Complexity

Yin-Tat Lee

Defining Efficiency

"Runs fast on typical real problem instances"

Pros:

- Sensible,
- Bottom-line oriented

Cons:

- Moving target (diff computers, programming languages)
- Highly subjective (how fast is "fast"? What is "typical"?)

Measuring Efficiency

Time ≈ # of instructions executed in a simple programming language

- only simple operations (+,*,-,=,if,call,…)
- each operation takes one time step
- each memory access takes one time step
- no fancy stuff (add these two matrices, copy this long string,…) built in

Time Complexity

Problem: An algorithm can have different running time on different inputs

Solution: The complexity of an algorithm associates a number **T(N)**, the "time" the algorithm takes on problem size **N**.

On which inputs of size **N**?

Mathematically,

T is a function that maps positive integers giving problem size to positive integers giving number of steps

Time Complexity (N)

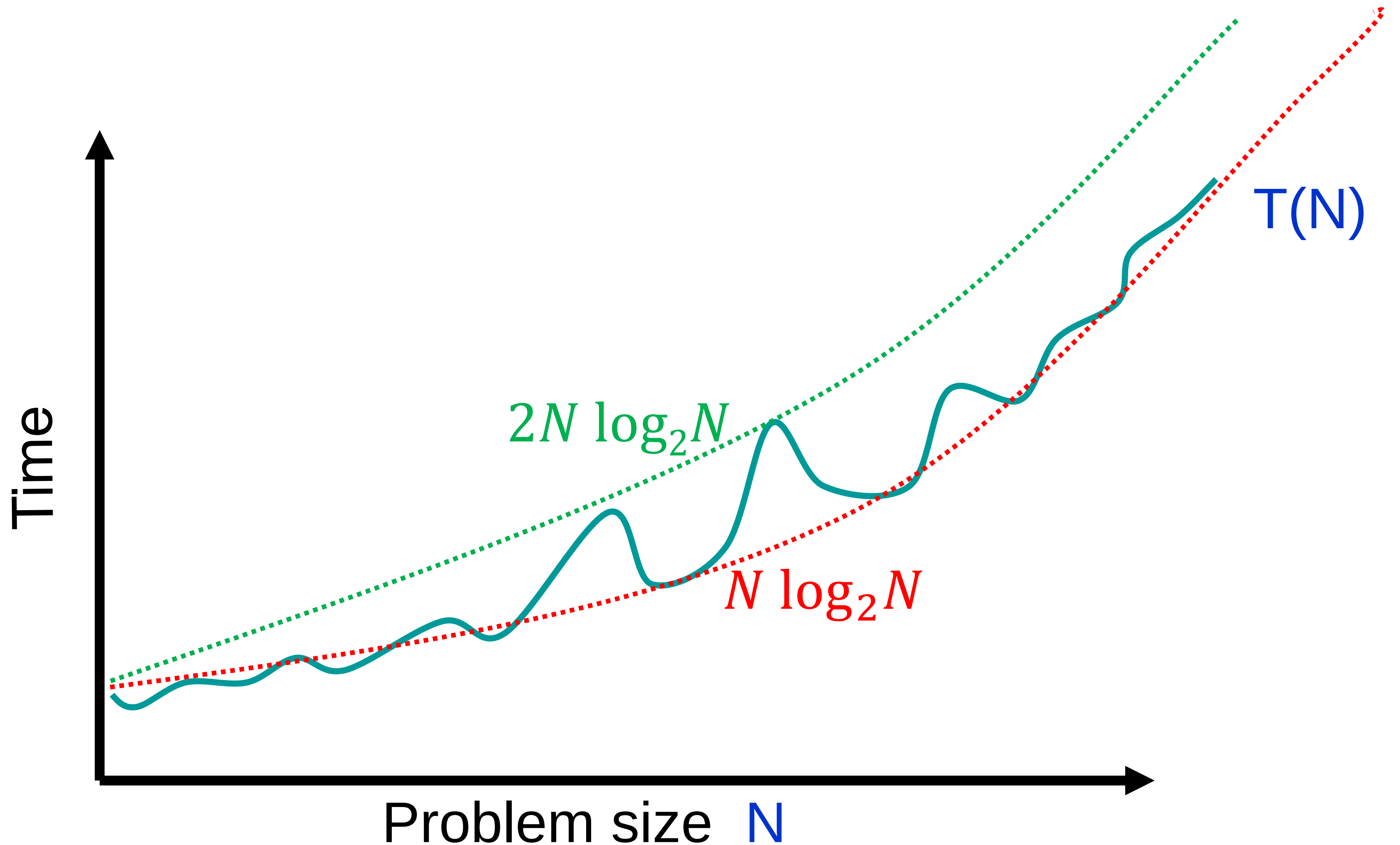
Worst Case Complexity: **max** # steps algorithm takes on any input of size **N**

This Couse

Average Case Complexity: **avg** # steps algorithm takes on inputs of size **N**

Best Case Complexity: **min** # steps algorithm takes on any input of size **N**

Time Complexity on Worst Case Inputs



O-Notation

Given two positive functions **f** and **g**

- **f(N)** is **O(g(N))** iff there is a constant **c>0** s.t.,
  f(N) is eventually always $\leq$ **c g(N)**

- **f(N)** is **Ω(g(N))** iff there is a constant **ε>0** s.t.,
  f(N) is $\geq$ **ε g(N)** for infinitely

- **f(N)** is **Θ(g(N))** iff there are constants c_1, c_2>0 so that
  eventually always $c_1 g(N) \leq f(N) \leq c_2 g(N)$

Asymptotic Bounds for common fns

- Polynomials:

  $a_0 + a_1 n + \cdots + a_d n^d$ is $O\left(n^d\right)$

- Logarithms:

  $\log_a n = O(\log_b n)$ for all constants $a, b > 0$

- Logarithms: log grows slower than every polynomial

  For all $x > 0$, $\log n = O(n^k)$

- $n \log n = O(n^{1.01})$

Running Time

An algorithm runs in polynomial time if $T(n) = n^{O(1)}$.

Equivalently, $T(n) = n^d$ for some constant d.

Name	Complexity class	Running time ($T(n)$)	Examples of running times	Example algorithms
constant time		$O(1)$	10	Determining if an integer (represented in binary) is even or odd
inverse Ackermann time		$O(\alpha(n))$		Amortized time per operation using a disjoint set
iterated logarithmic time		$O(\log^* n)$		Distributed coloring of cycles
log-logarithmic		$O(\log \log n)$		Amortized time per operation using a bounded priority queue[2]
logarithmic time	DLOGTIME	$O(\log n)$	$\log n$, $\log(n^2)$	Binary search
polylogarithmic time		$\mathrm{poly}(\log n)$	$(\log n)^2$	
fractional power		$O(n^c)$ where $0 < c < 1$	$n^{1/2}$, $n^{2/3}$	Searching in a kd-tree
linear time		$O(n)$	n	Finding the smallest or largest item in an unsorted array
"n log star n" time		$O(n \log^* n)$		Seidel's polygon triangulation algorithm.
quasilinear time		$O(n \log n)$	$n \log n$, $\log n!$	Fastest possible comparison sort; Fast Fourier transform.
quadratic time		$O(n^2)$	n^2	Bubble sort; Insertion sort; Direct convolution Stable matching!
cubic time		$O(n^3)$	n^3	Naive multiplication of two $n \times n$ matrices. Calculating partial correlation.
polynomial time	P	$2^{O(\log n)} = \mathrm{poly}(n)$	n, $n \log n$, n^{10}	Karmarkar's algorithm for linear programming; AKS primality test
quasi-polynomial time	QP	$2^{\mathrm{poly}(\log n)}$	$n^{\log \log n}$, $n^{\log n}$	Best-known $O(\log^2 n)$-approximation algorithm for the directed Steiner tree problem.
sub-exponential time (first definition)	SUBEXP	$O(2^{n^\varepsilon})$ for all $\varepsilon > 0$	$O(2^{\log n^{\log \log n}})$	Assuming complexity theoretic conjectures, BPP is contained in SUBEXP.[3]
sub-exponential time (second definition)		$2^{o(n)}$	$2^{n^{1/3}}$	Best-known algorithm for integer factorization ~~and graph isomorphism~~
exponential time (with linear exponent)	E	$2^{O(n)}$	1.1^n, 10^n	Solving the traveling salesman problem using dynamic programming
exponential time	EXPTIME	$2^{\mathrm{poly}(n)}$	2^n, 2^{n^2}	Solving matrix chain multiplication via brute-force search
factorial time		$O(n!)$	$n!$	Solving the traveling salesman problem via brute-force search
double exponential time	2-EXPTIME	$2^{2^{\mathrm{poly}(n)}}$	2^{2^n}	Deciding the truth of a given statement in Presburger arithmetic

Why it matters?

Suppose we can do 1 million operations per second.

	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

not only get very big, but do so *abruptly*, which likely yields erratic performance on small instances

Outdated: Nvidia announced a “computer” this Tue that do 2 quadrillion (2×10^{15}) operations/sec.
It brings down the 31,710 years to 500 sec.
However, 2^{100} operations still takes millions of years.

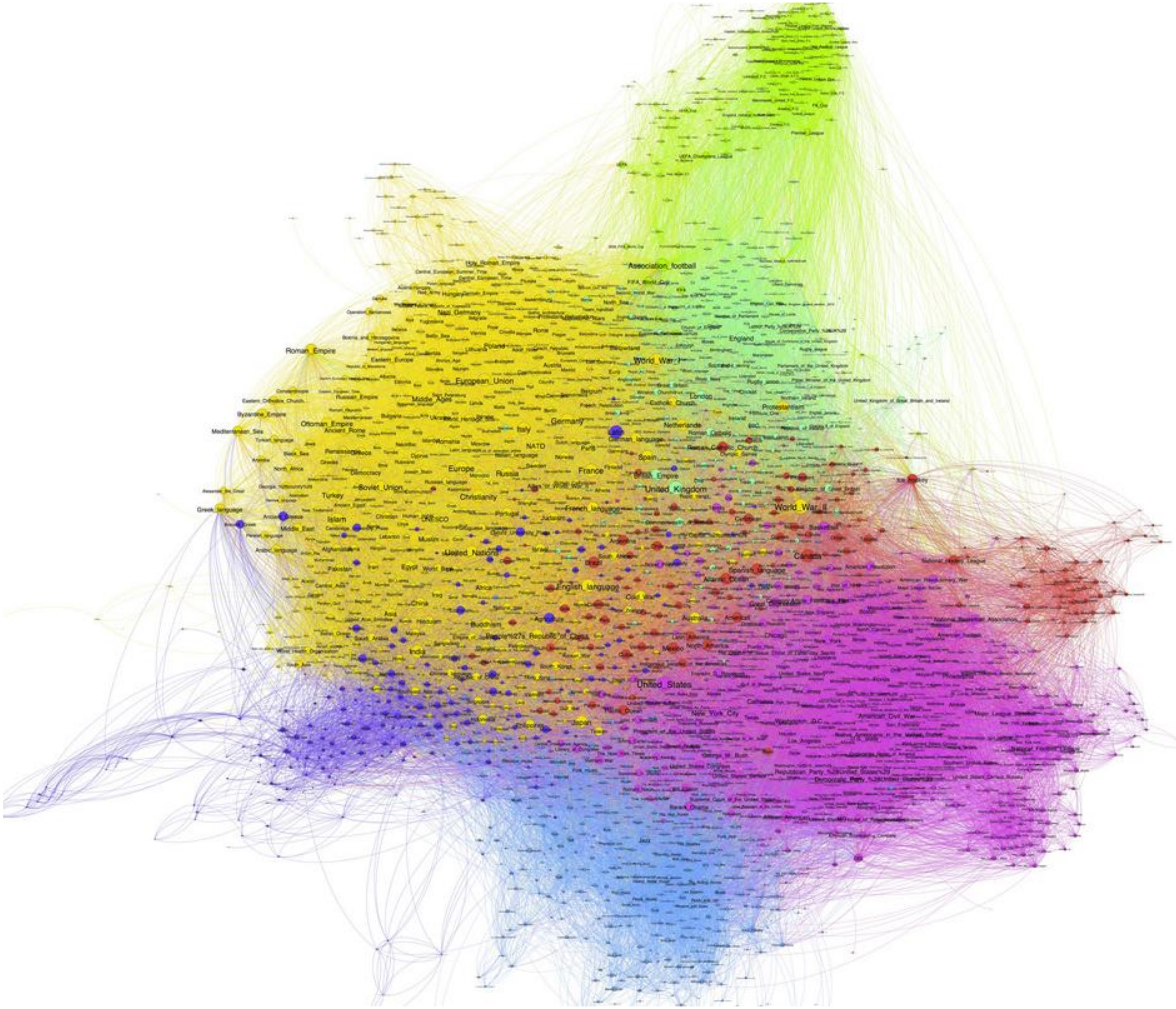
Why “Polynomial”?

Point is not that n^{2000} is a practical bound, or that the differences among n and 2n and n^2 are negligible.

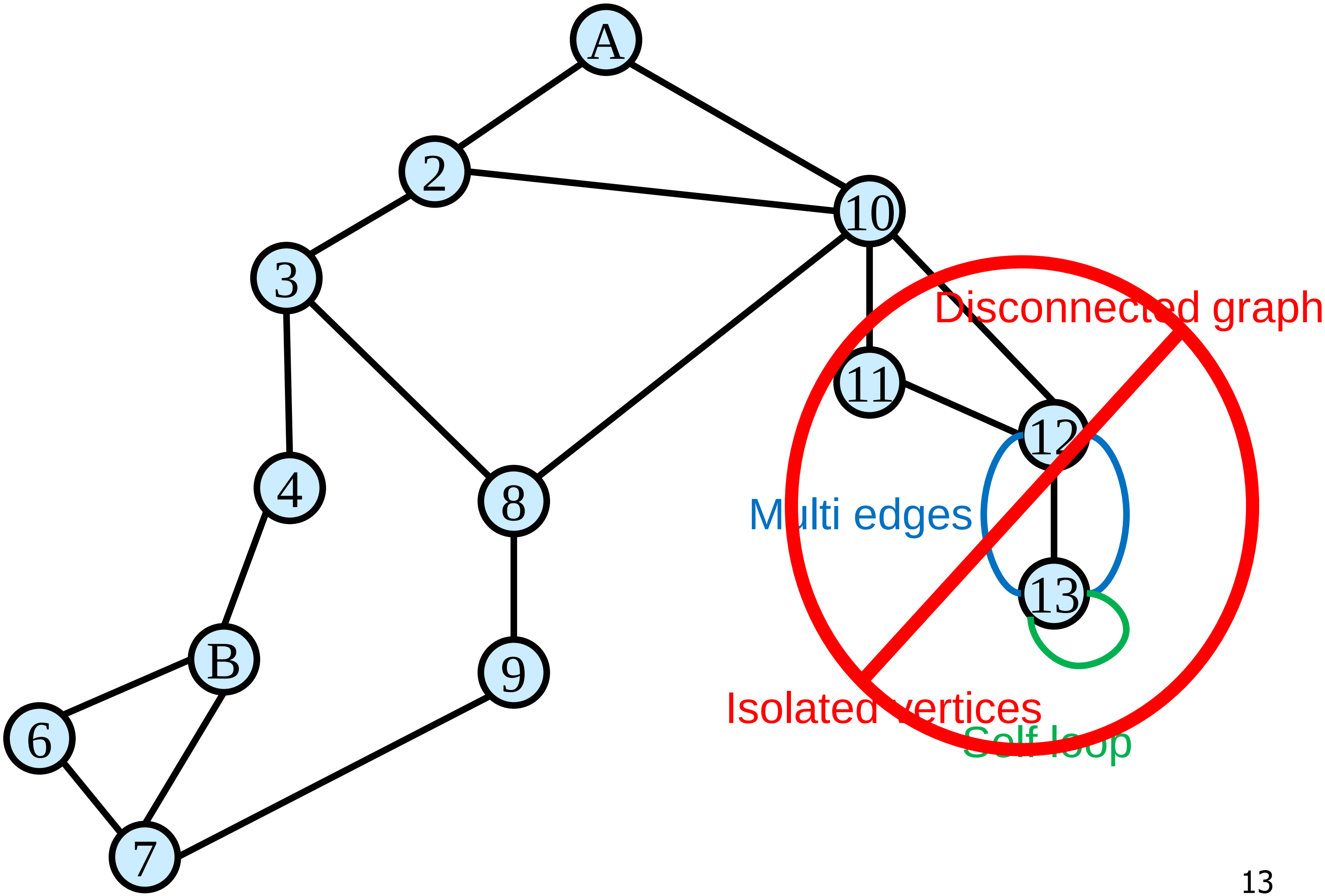
Rather, simple theoretical tools may not easily capture such differences, whereas exponentials are qualitatively different from polynomials, so more amenable to theoretical analysis.

- “My problem is in P” is a starting point for a more detailed analysis
- “My problem is not in P” may suggest that you need to shift to a more tractable variant

Graphs


Association football
Roman Empire
European Union
World War I
Germany
Europe
Russia
France
Soviet Union
Christianity
Islam
United Kingdom
World War II
Canada
English language
United States
New York City
Japan

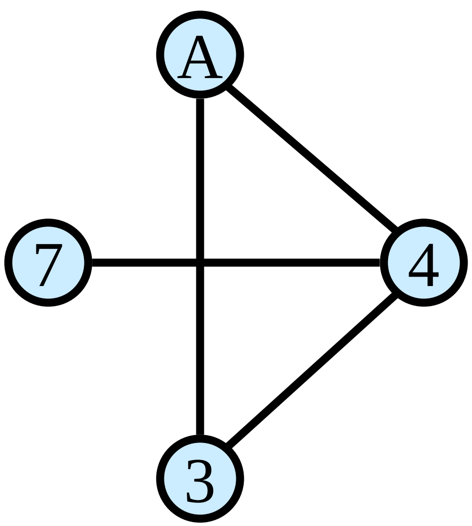
Undirected Graphs G=(V,E)

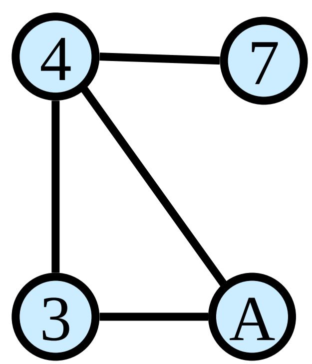


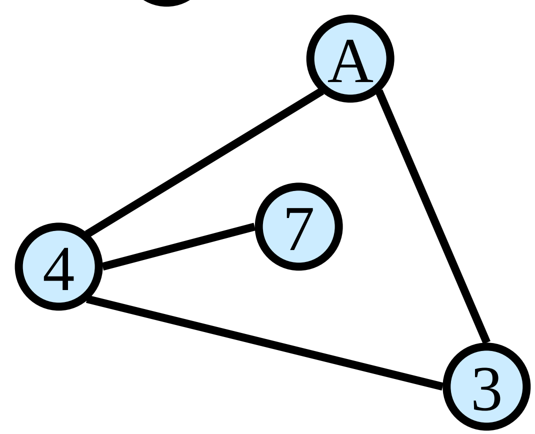
Graphs don’t Live in Flat Land

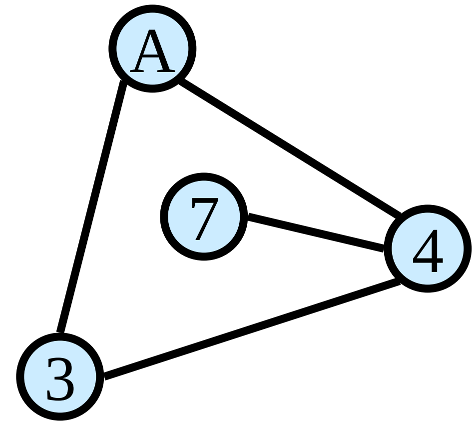
Geometrical drawing is mentally convenient, but mathematically irrelevant:

4 drawings of a single graph:

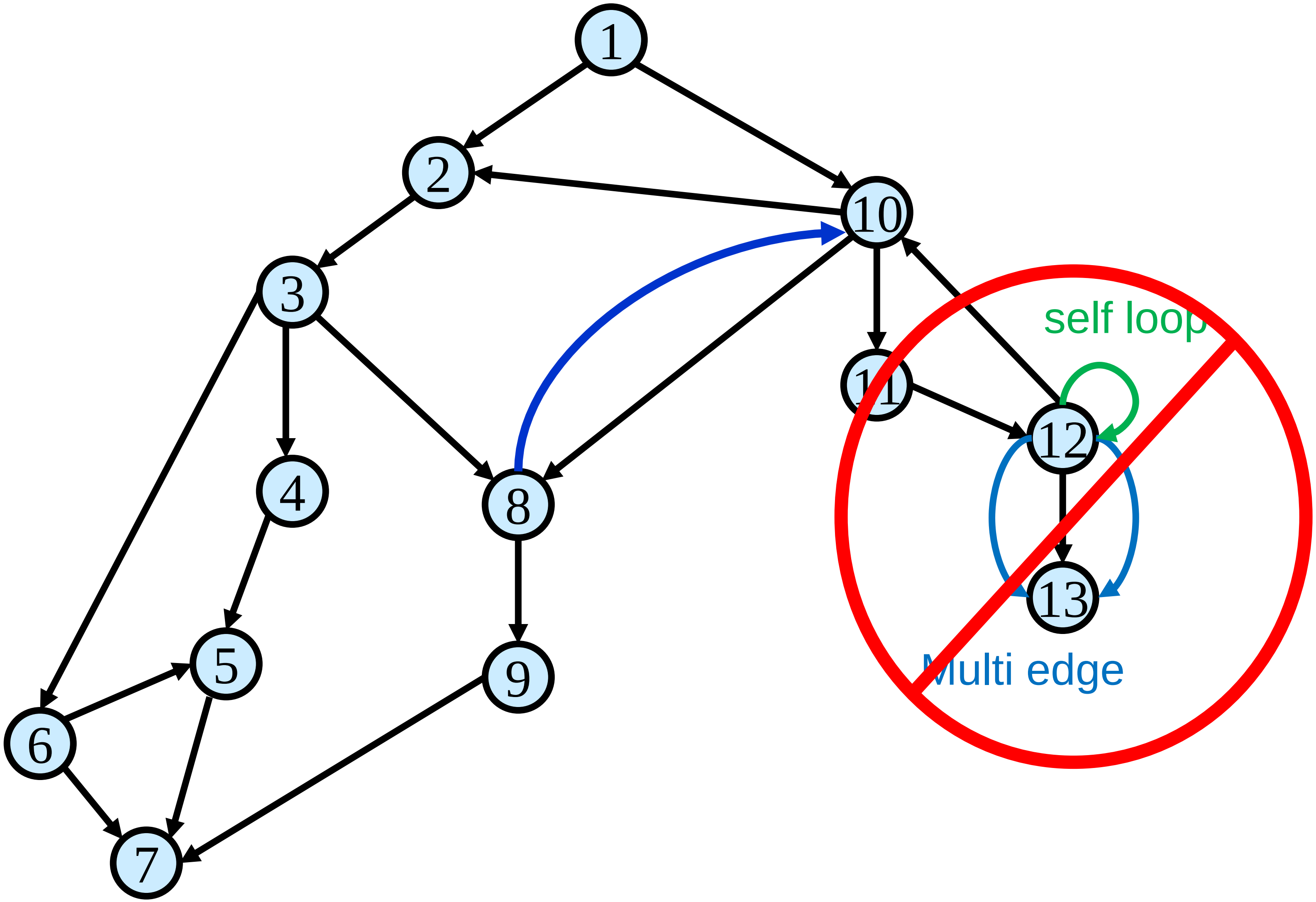






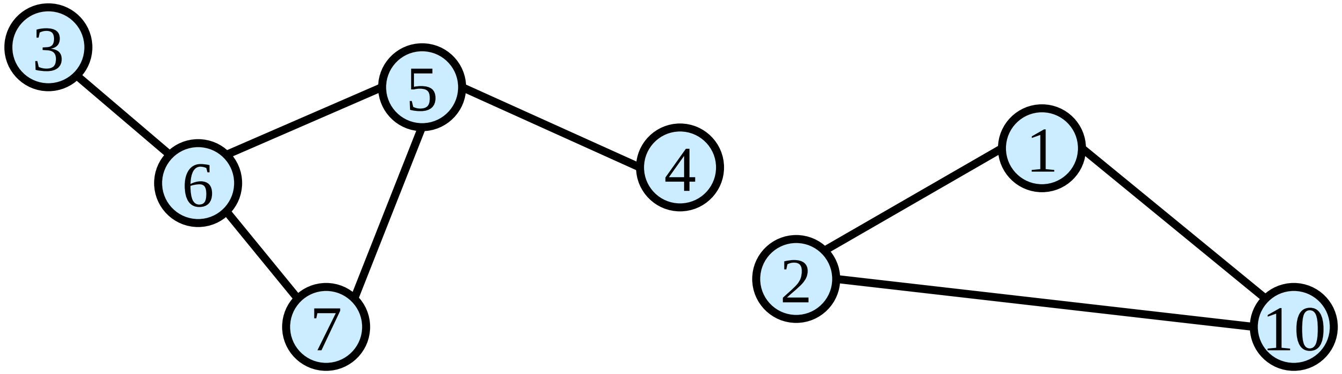


Directed Graphs



Terminology

- Degree of a vertex: # edges that touch that vertex



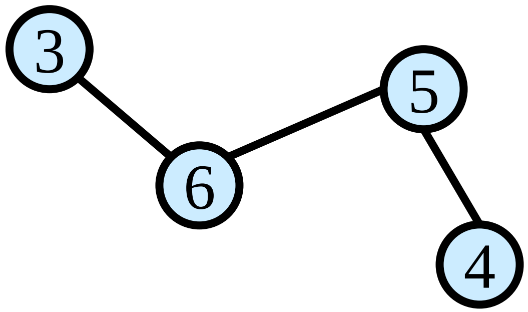
deg(6)=3

- Connected: Graph is connected if there is a path between every two vertices

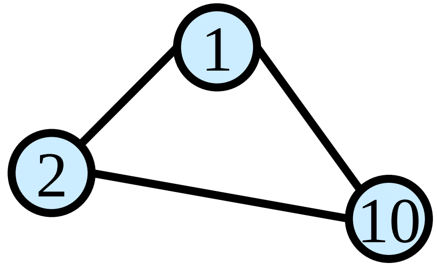
- Connected component: Maximal set of connected vertices

Terminology (cont'd)

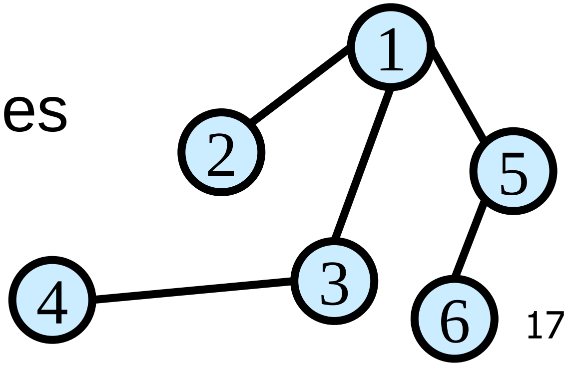
- Path: A sequence of distinct vertices s.t. each vertex is connected to the next vertex with an edge



- Cycle: Path of length > 2 that has the same start and end



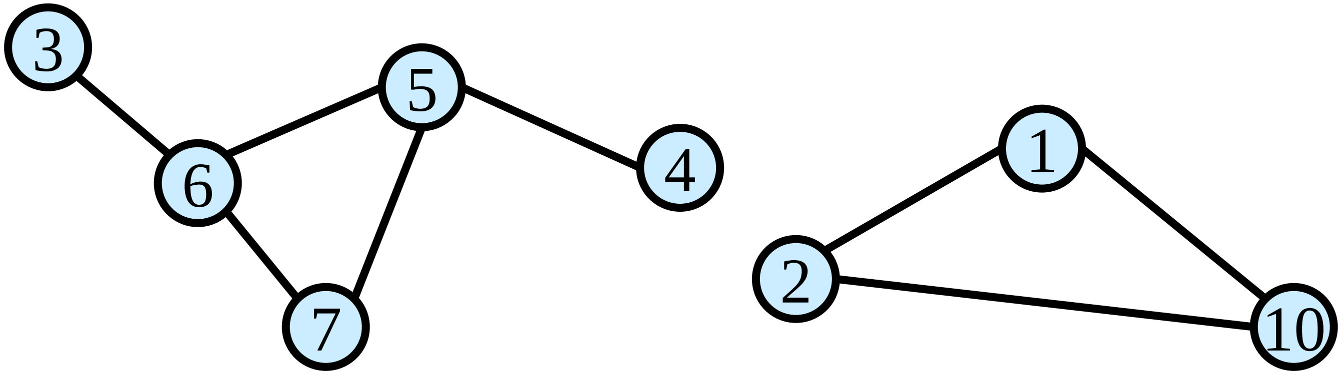
- Tree: A connected graph with no cycles



Degree Sum

Claim: In any undirected graph, the number of edges is equal to $(1/2)\sum_{\text{vertex } v} \deg(v)$

Pf: $\sum_{\text{vertex } v} \deg(v)$ counts every edge of the graph exactly twice; once from each end of the edge.



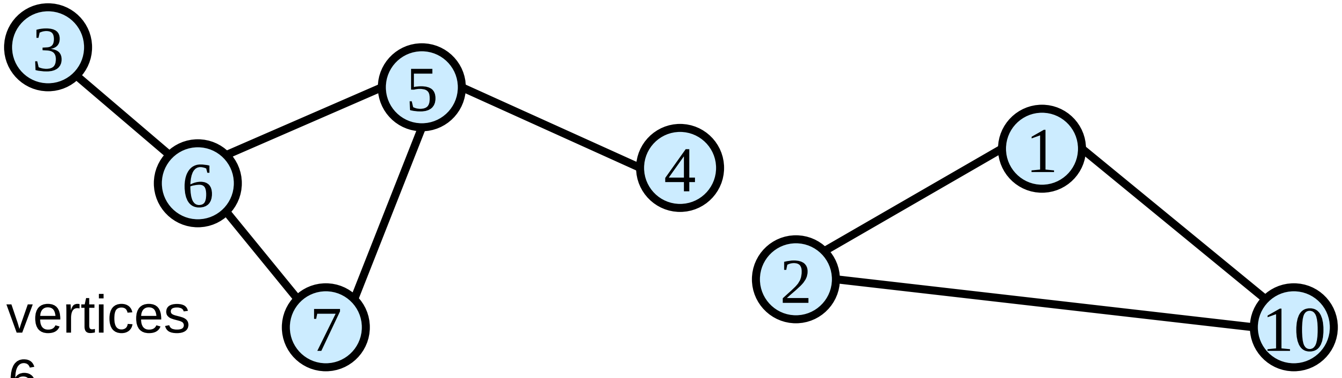
|E|=8

$$\sum_{\text{vertex } v} \deg(v) = 2 + 2 + 1 + 1 + 3 + 2 + 3 + 2 = 16$$

Odd Degree Vertices

Claim: In any undirected graph, the number of odd degree vertices is even

Pf: In previous claim we showed sum of all vertex degrees is even. So there must be even number of odd degree vertices, because sum of odd number of odd numbers is odd.



4 odd degree vertices
3, 4, 5, 6

#edges

Let $G = (V, E)$ be a graph with $n = |V|$ vertices and $m = |E|$ edges.

Claim: $0 \leq m \leq \binom{n}{2} = \frac{n(n-1)}{2} = O(n^2)$

Pf: Since every edge connects two distinct vertices (i.e., G has no loops)

and no two edges connect the same pair of vertices (i.e., G has no multi-edges)

It has at most $\binom{n}{2}$ edges.