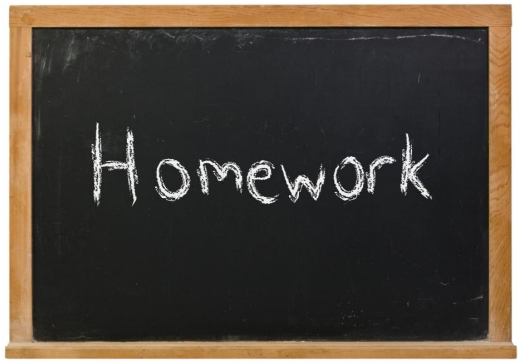
CSE 421: Introduction to Algorithms

Course Overview

Shayan Oveis Gharan

Administrativia Stuffs



HW1 is due Thursday April 07 at 11:59PM
Please submit to Gradescope

Late Submission: Fill out an extension request in edstem.

How to submit?

- Double check your submission before the deadline!!
- Please typeset your solution if possible

Guidelines:

- Always justify your answer
- You can collaborate, but you must write solutions on your own
- Your proofs should be clear, well-organized, and concise. Spell out main idea.
- Sanity Check: Spell out when you use assumptions of the problem

Induction: Intro 2

Prove that if n+1 balls are placed into n bins then one bin has at least two balls.

Def: P(n): If n+1 balls are placed into n bins then one bin has at least two balls.

Base Case: P(1) holds. Two balls into one bin

IH: P(n-1) holds for some $n \geq 2$

IS: Goal is to prove P(n). Suppose n+1 balls are placed into n bins. Need to show a bin has ≥ 2 balls. Look at bin 1.

Case 1: Bin 1 has at least two balls. Then we are done.

Case 2: Bin 1 has 1 ball. Then. we have placed n balls into bins 2,..,n. So, by IH one bin has at least two balls.

Case 3: Bin 1 has 0 balls. Remove an arbitrary ball. Then, we have n balls in bins 2,..,n. So, by IH a bin has ≥ 2 balls

Main Objective: Design Efficient Algorithms that finds optimum solutions in the Worst Case

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; write it/charge for it as above

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**

Why Worst-case Inputs?

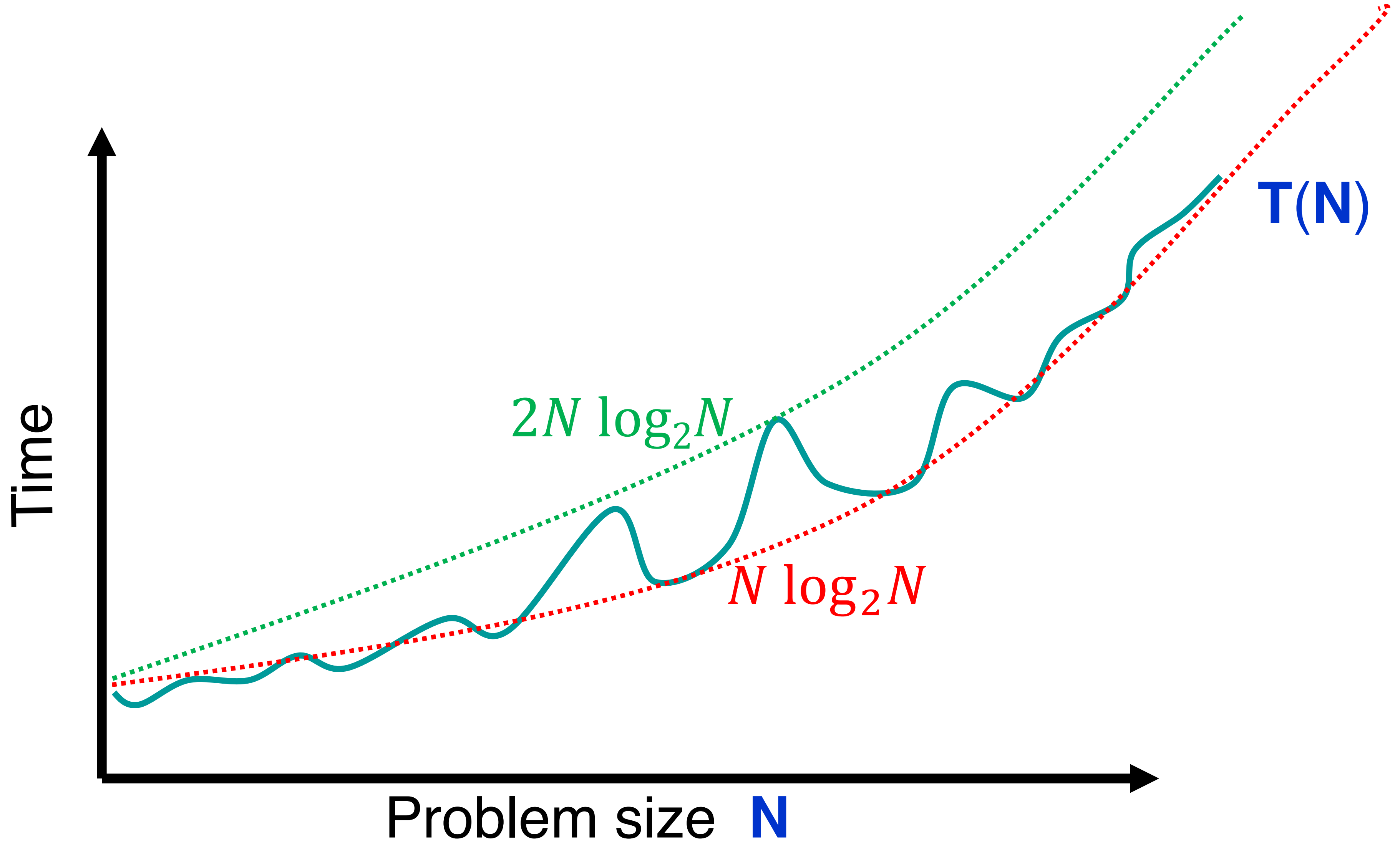
- Analysis is typically easier
- Useful in real-time applications

  e.g., space shuttle, nuclear reactors)
- Worst-case instances kick in when an algorithm is run as a module many times

  e.g., geometry or linear algebra library
- Useful when running competitions

  e.g., airline prices
- Unlike average-case no debate about the right definition

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 $\varepsilon > 0$ s.t.,
  f(N) is $\geq \varepsilon$ **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 $O(n^d)$

- Logarithms:

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

- Logarithms: log grows slower than every polynomial

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

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

Efficient = Polynomial Time

An algorithm runs in polynomial time if $T(n)=O(n^d)$ for some constant d independent of the input size n.

Why Polynomial time?

If problem size grows by at most a constant factor then so does the running time

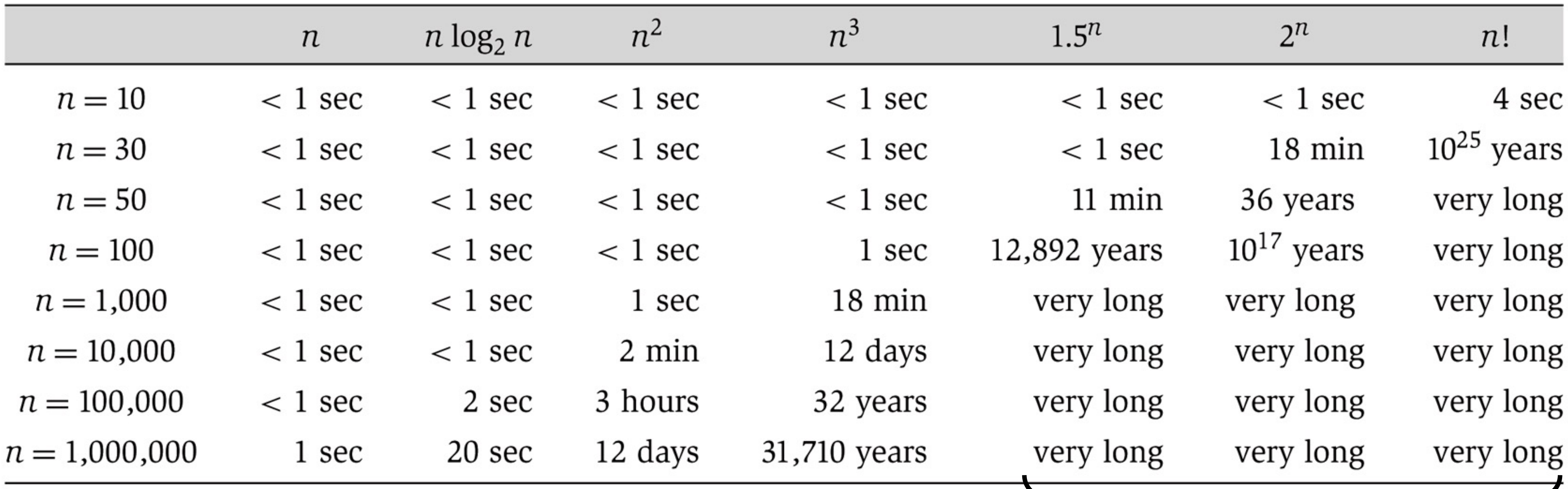
- E.g. **$T(2N) \leq c(2N)^k \leq 2^k(cN^k)$**
- Polynomial-time is exactly the set of running times that have this property

Typical running times are small degree polynomials, mostly less than **N^3**, at worst **N^6**, not **N^{100}**

Why it matters?

- #atoms in universe < 2^{240}
- Life of the universe < 2^{54} seconds
- A CPU does < 2^{30} operations a second

If every atom is a CPU, a 2^n time ALG cannot solve n=350 if we start at Big-Bang.



	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

Why “Polynomial”?

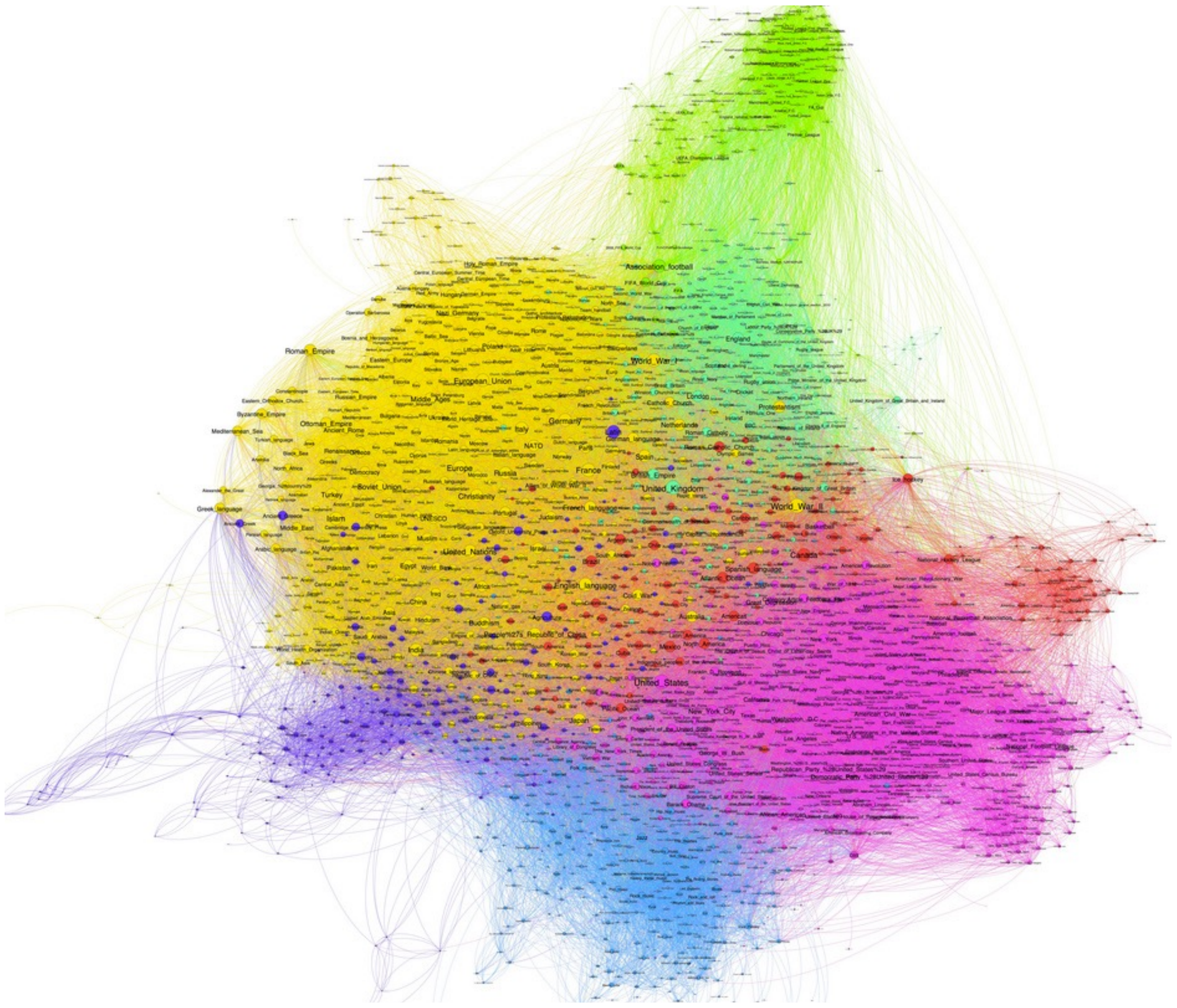
Point is not that n^{2000} is a practical bound, or that the differences among n and 2n and n^2 are negligible.

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

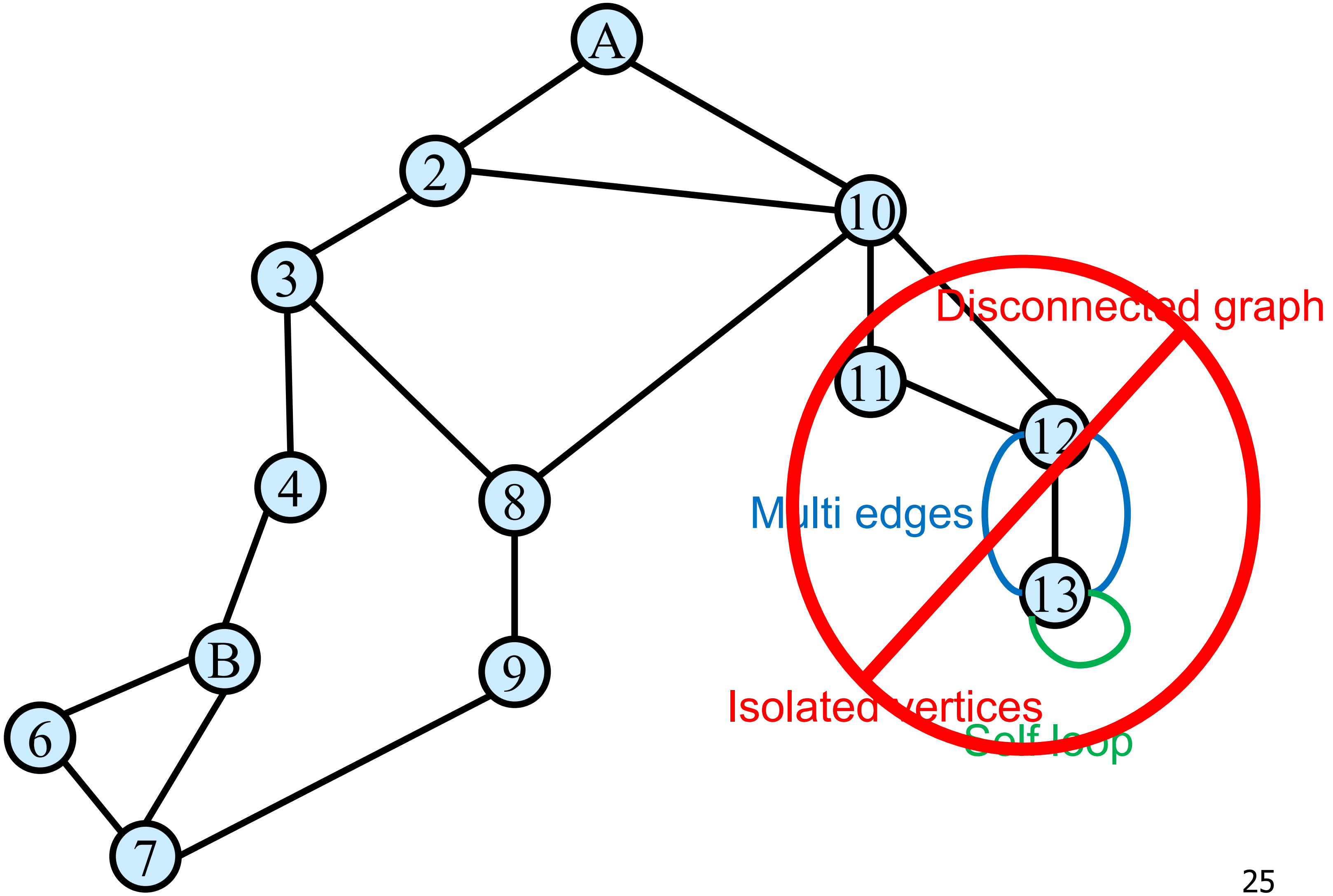
- “My problem is in P” is a starting point for a more detailed analysis
- “My problem is not in P” may suggest that you need to shift to a more tractable variant

Graphs

Graphs


Association_football
Roman_Empire
European_Union
Germany
World_War_I
World_War_II
United_Kingdom
Canada
United_Nations
English_language
United_States
Europe
Russia
France
India
Japan
Ice_hockey
New_York_City
Washington,_D.C.

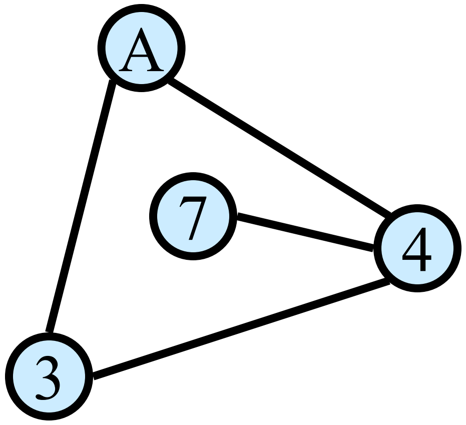
Undirected Graphs G=(V,E)

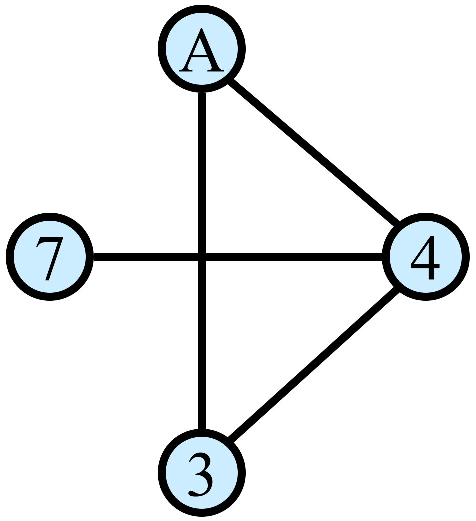


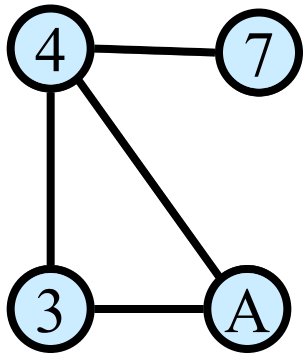
Graphs don’t Live in Flat Land

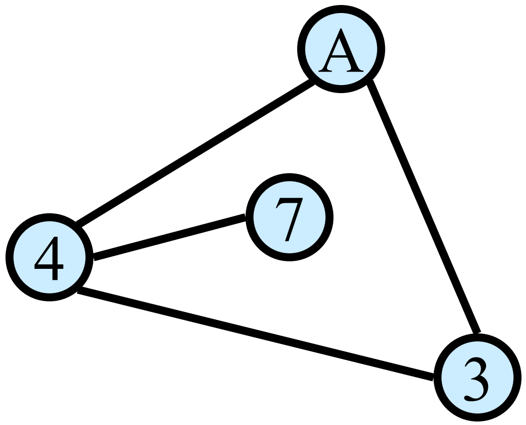
Geometrical drawing is mentally convenient, but mathematically irrelevant:

4 drawings of a single graph:

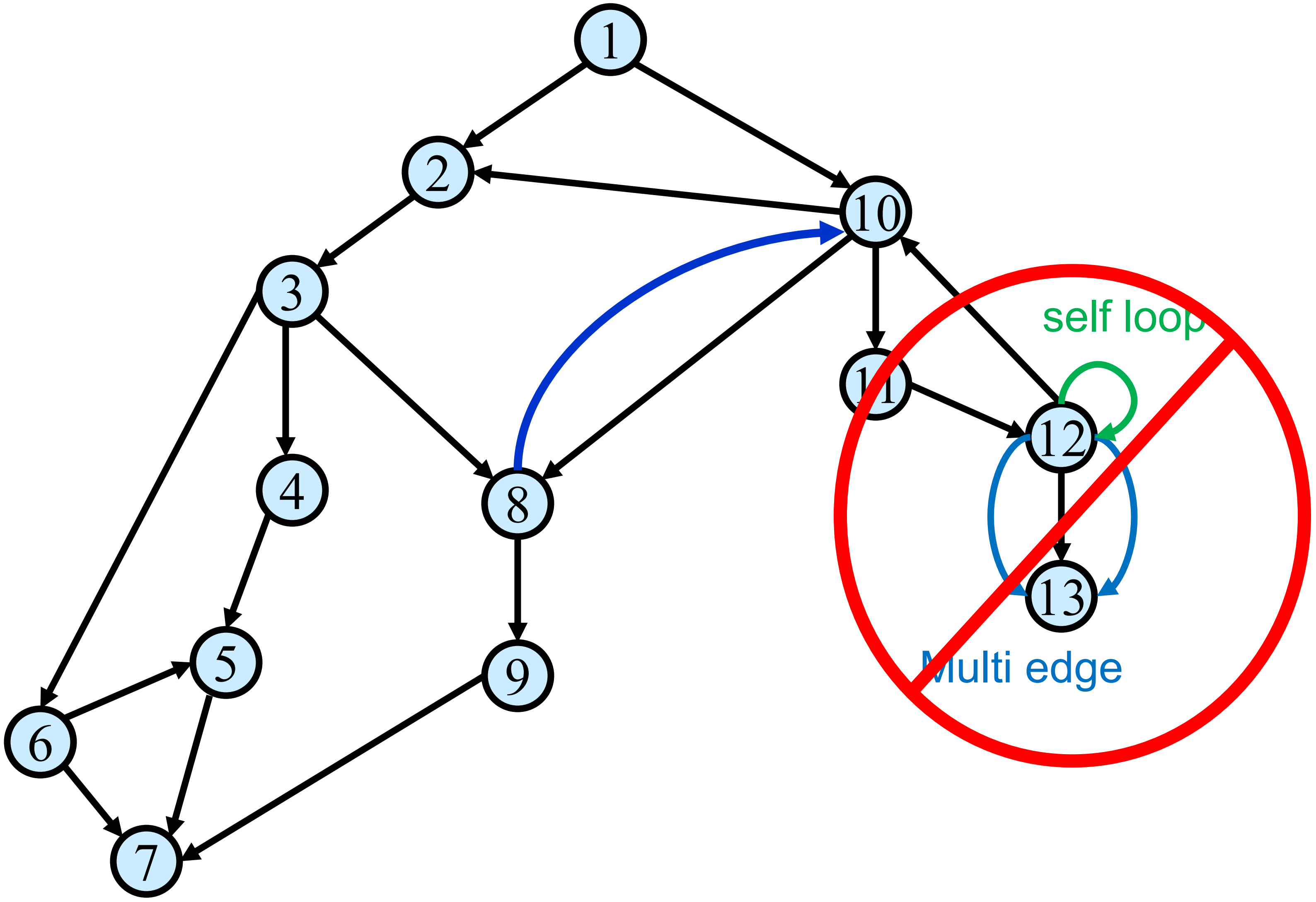








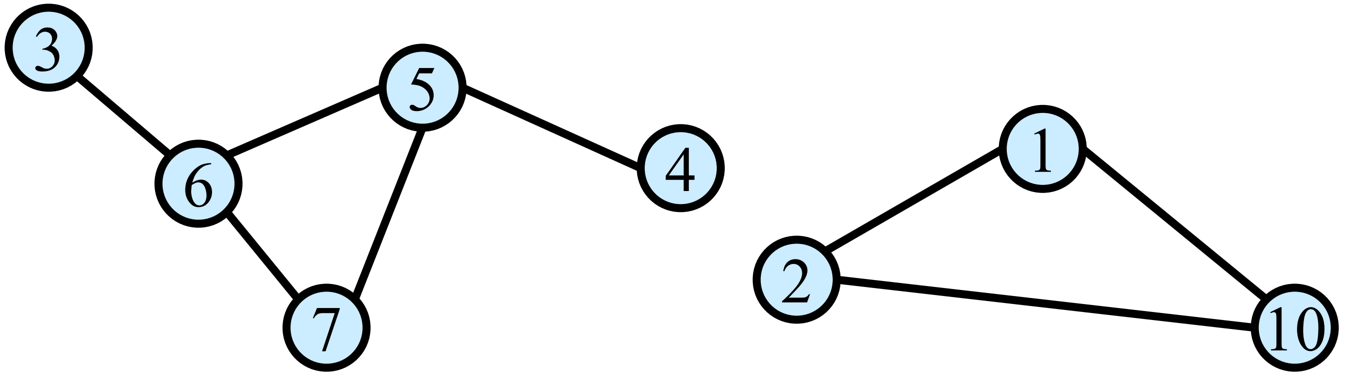
Directed Graphs



Terminology

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

deg(6)=3

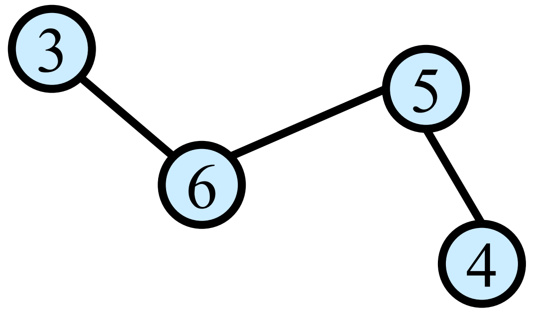


- Connected: Graph is connected if there is a path between every two vertices

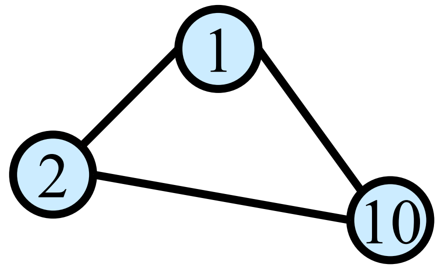
- Connected component: Maximal set of connected vertices

Terminology (cont'd)

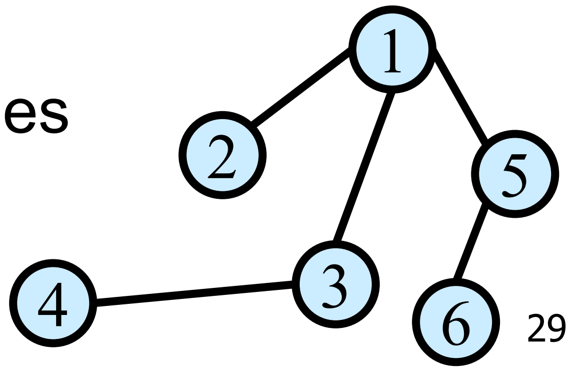
- Path: A sequence of distinct vertices s.t. each vertex is connected to the next vertex with an edge



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



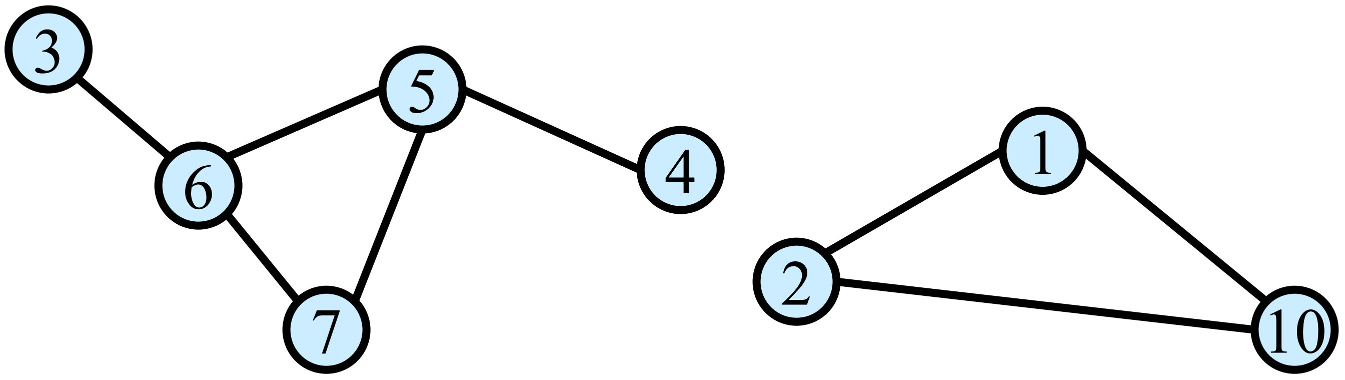
- Tree: A connected graph with no cycles



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.



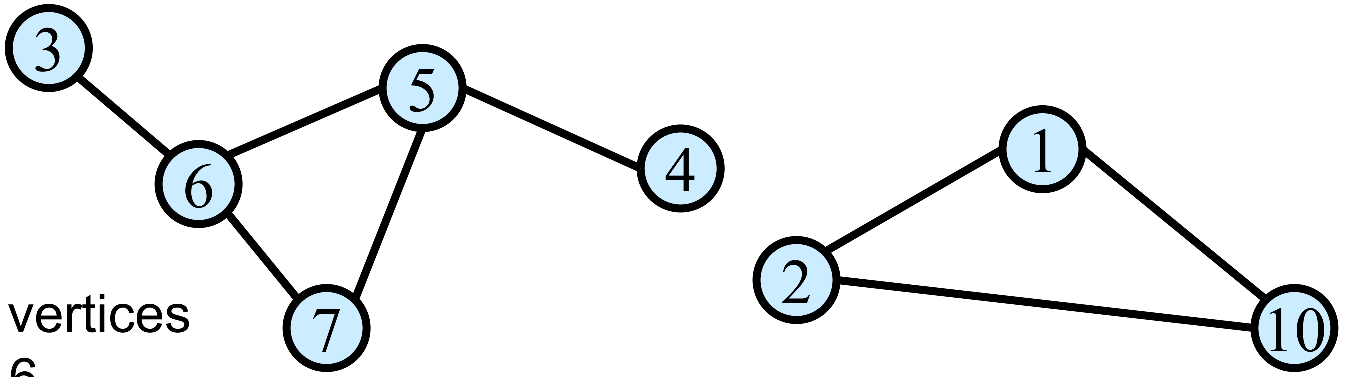
|E|=8

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

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

Degree 1 vertices

Claim: If G has no cycle, then it has a vertex of degree ≤ 1

(So, every tree has a leaf)

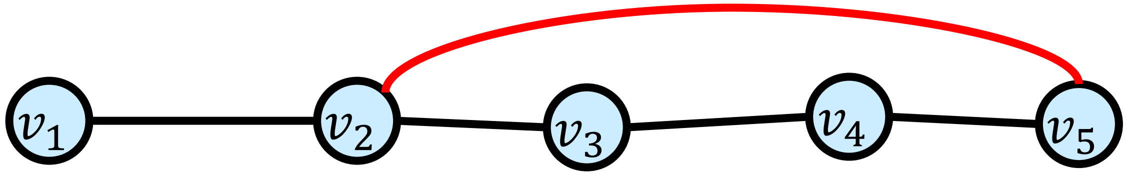
Pf: (By contradiction)

Suppose every vertex has degree ≥ 2.

Start from a vertex v_1 and follow a path, $v_1, \dots, v_i$ when we are at v_i we choose the next vertex to be different from v_{i-1}. We can do so because $\deg(v_i) \geq 2$.

The first time that we see a repeated vertex $(v_j = v_i)$ we get a cycle.

We always get a repeated vertex because G has finitely many vertices



Trees and Induction

Claim: Show that every tree with n vertices has n-1 edges.

Pf: By induction.

Base Case: n=1, the tree has no edge

IH: Suppose every tree with n-1 vertices has n-2 edges

IS: Let T be a tree with n vertices.

So, T has a vertex v of degree 1.

Remove v and the neighboring edge, and let T' be the new graph.

We claim T' is a tree: It has no cycle, and it must be connected.

So, T' has n-2 edges and T has n-1 edges.