Asymptotic order of growth

The order of growth is a notation so that we can argue about running times.

- it's useful to think of the running time as a mathematical function of n

example: T(n) = n^2+3n+1 *is* $O(n^2)$, $T(n) = 1.5^{n-1}+2n$ *is* $O(1.5^n)$

"asymptotic" — we compare the exact running time of an algorithm (or magnitude of a function) to the most simple function with similar growth.

"growth" — how the number of computational steps is increasing as the input size *n* grows.

When is an algorithm efficient?

Good algorithm: works in practice! - what does that mean?

- algorithm is correct
- efficient: runs reasonable fast

How to define 'reasonable fast'?

Desirable scaling property: When the input size doubles, the running time should increase by at most some constant factor C.

(mock) TopHat question

Desirable scaling property. When the input size doubles, the running time should increase by at most some constant factor C.

Which of these functions scale nicely?

$T(n) = n^3 + n^2$. Then $T(2n) =$

$T(n) = 3^n + n^2$. Then $T(2n) =$

$T(n) = n!$. Then $T(2n) =$

When is an algorithm efficient?

An algorithm is polynomial if there exist constants c and d, such that for *any* input size n the running time of the algorithm is at most cn^d.

Example of polynomial running times:

$$T(n) = n^2 + 2n^5 - 3n^3$$

$$T(n) = 3n \log n + n^4 + 2$$

$$T(n) = \log n$$

Example of exponential running times:

$$T(n) = 5 \cdot 2^n + n^3$$

$$T(n) = n!$$

We consider polynomial algorithms to be “efficient” , exponential algorithms to be “infeasible”

- in practice we need polynomial algorithms with low exponents
- brute-force algorithms tend to be exponential

Asymptotic running time of an algorithm

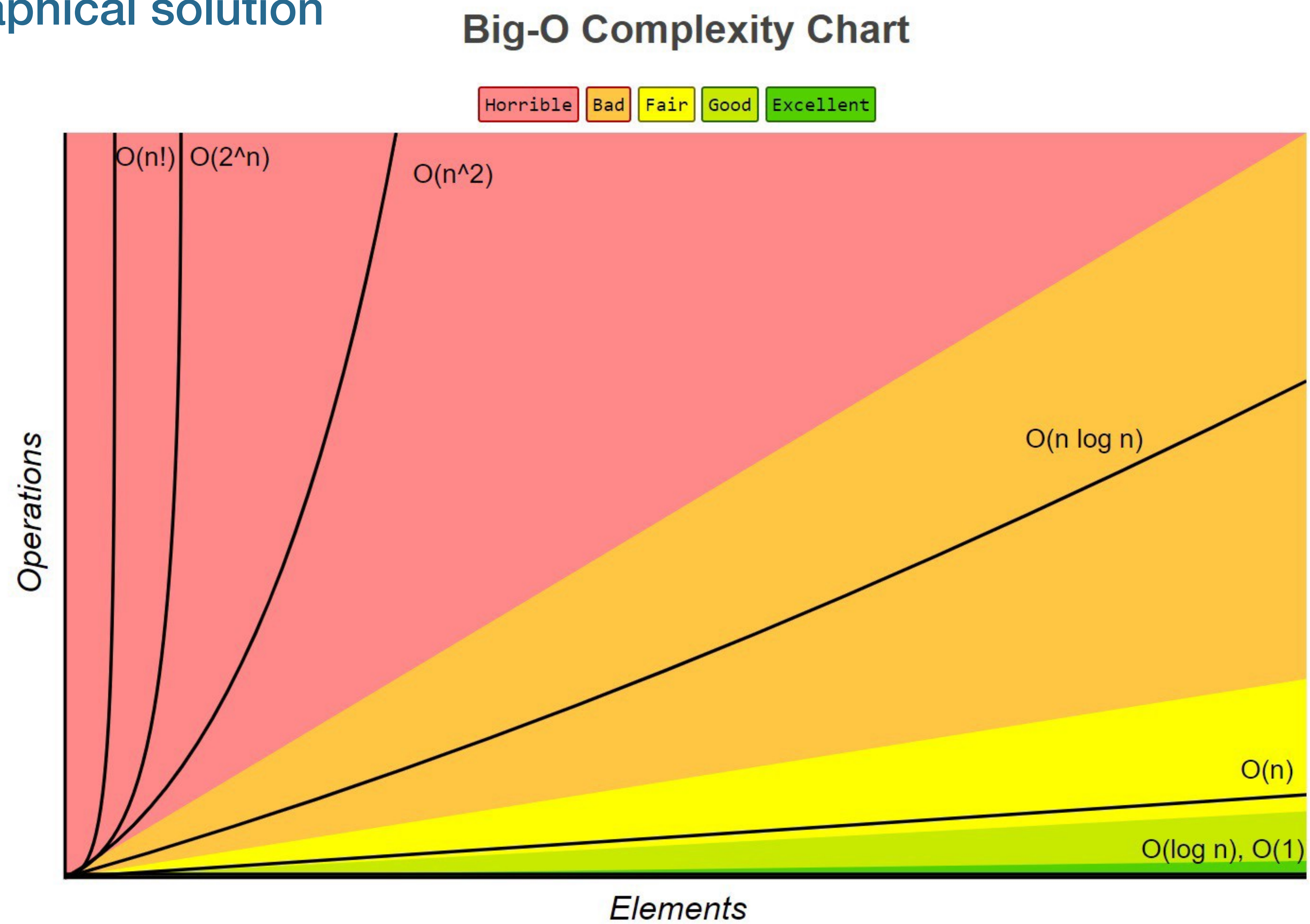
Asymptotic running time: An approximation of the number of computational steps performed by an algorithm by a "simple" function of similar order of growth.

- it is always expressed as a function of the input size

Goal for today is to define the

- asymptotic upper bound — big-Oh $O(\ )$
- asymptotic lower bound — big-Omega $\Omega(\ )$
- asymptotic (tight) bound — big-Theta $\Theta(\ )$

Graphical solution



source: https://towardsdatascience.com/understanding-time-complexity-with-python-examples-2bda6e8158a7

Graphical solution

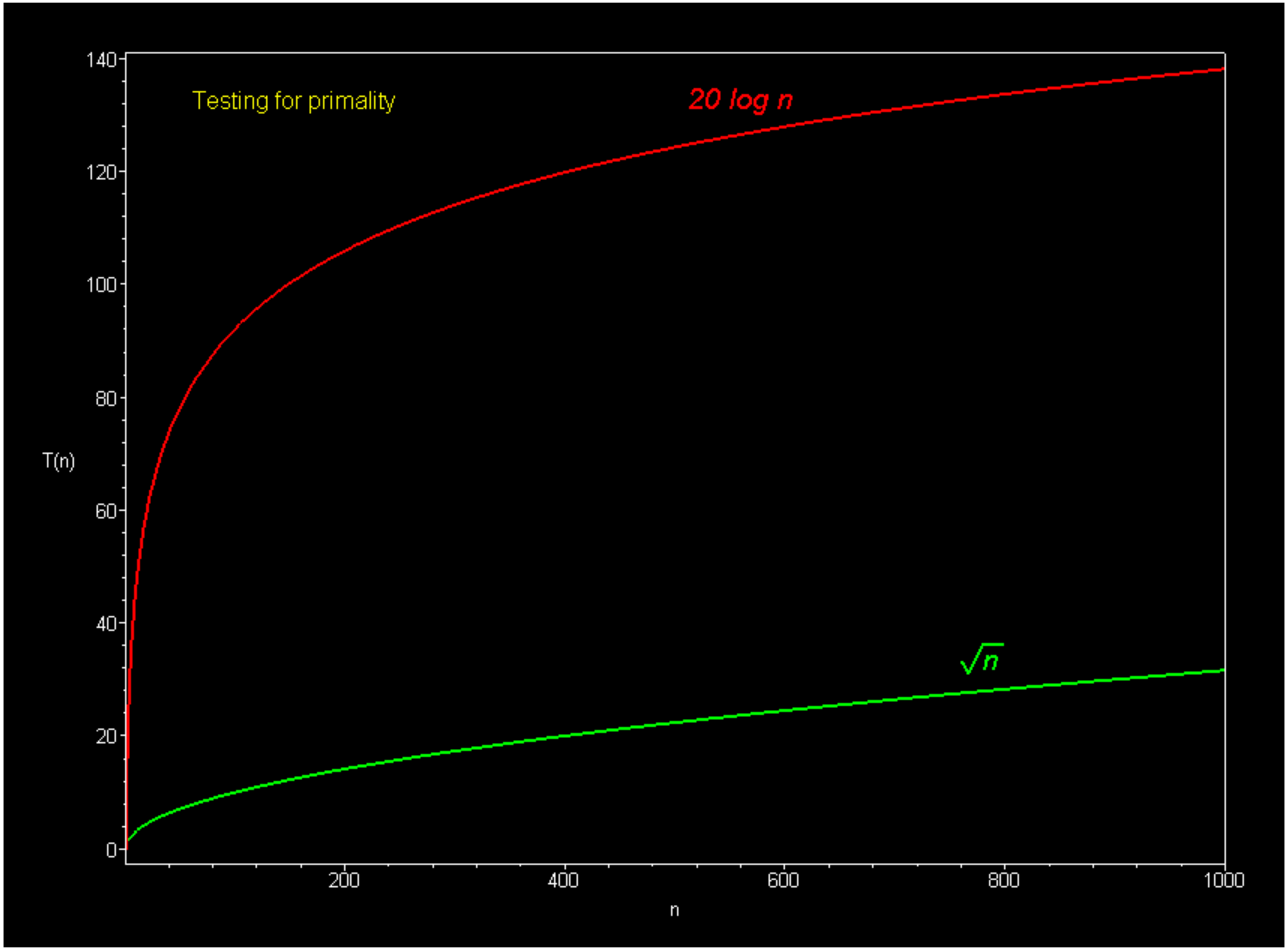


image source: http://science.slc.edu/~jmarshall/courses/2006/fall/accelcs/pub/week15/BigO/

Graphical solution

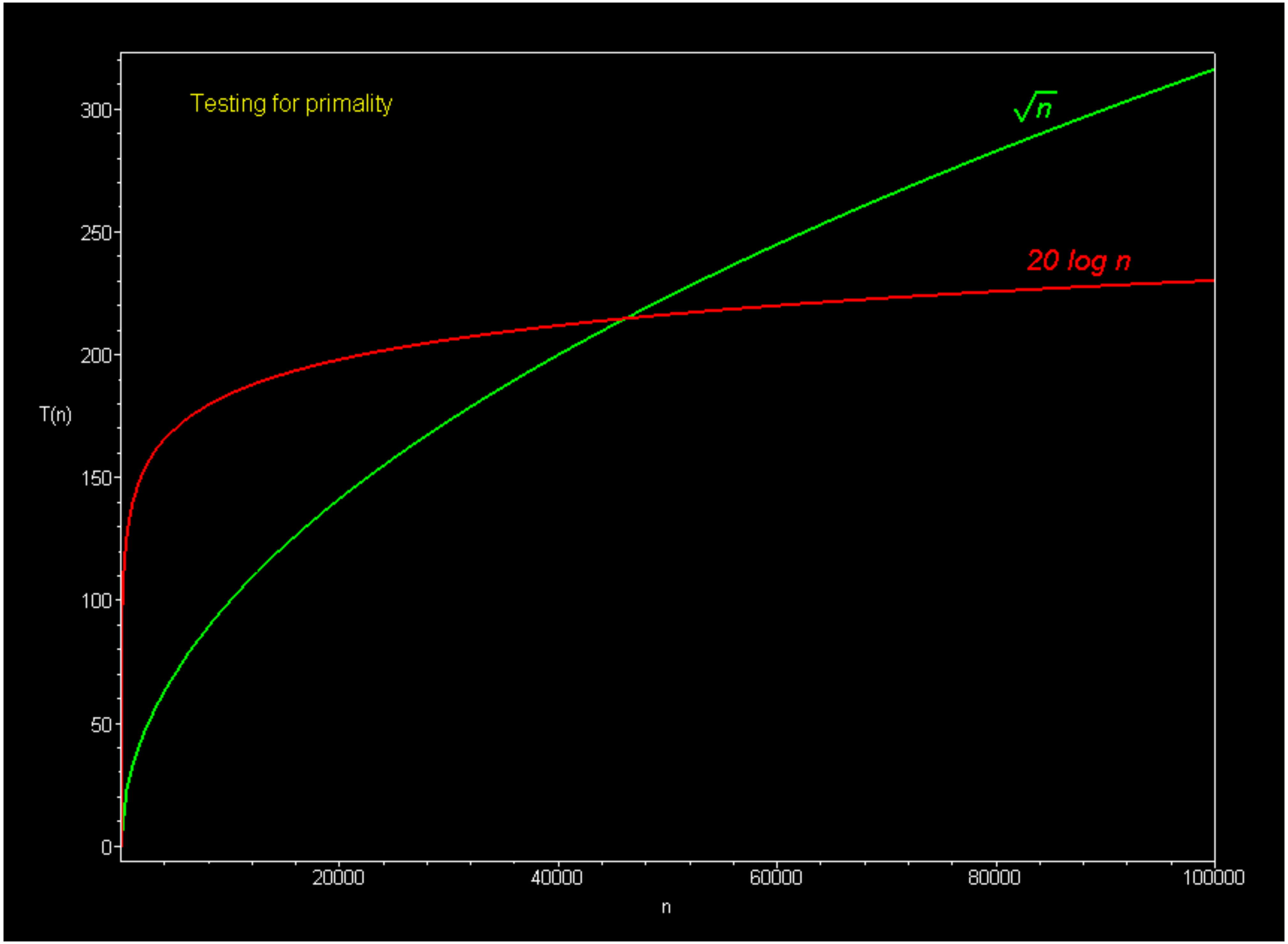
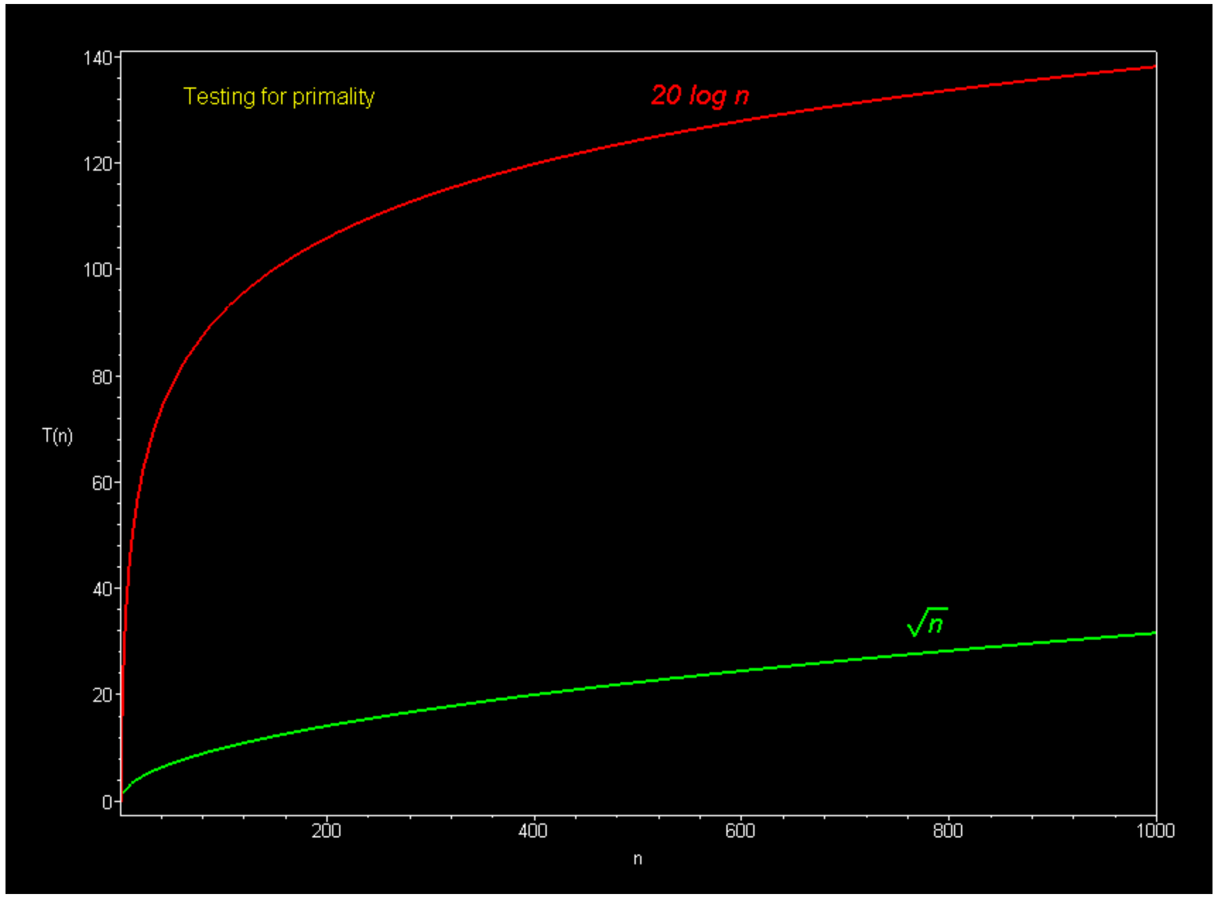
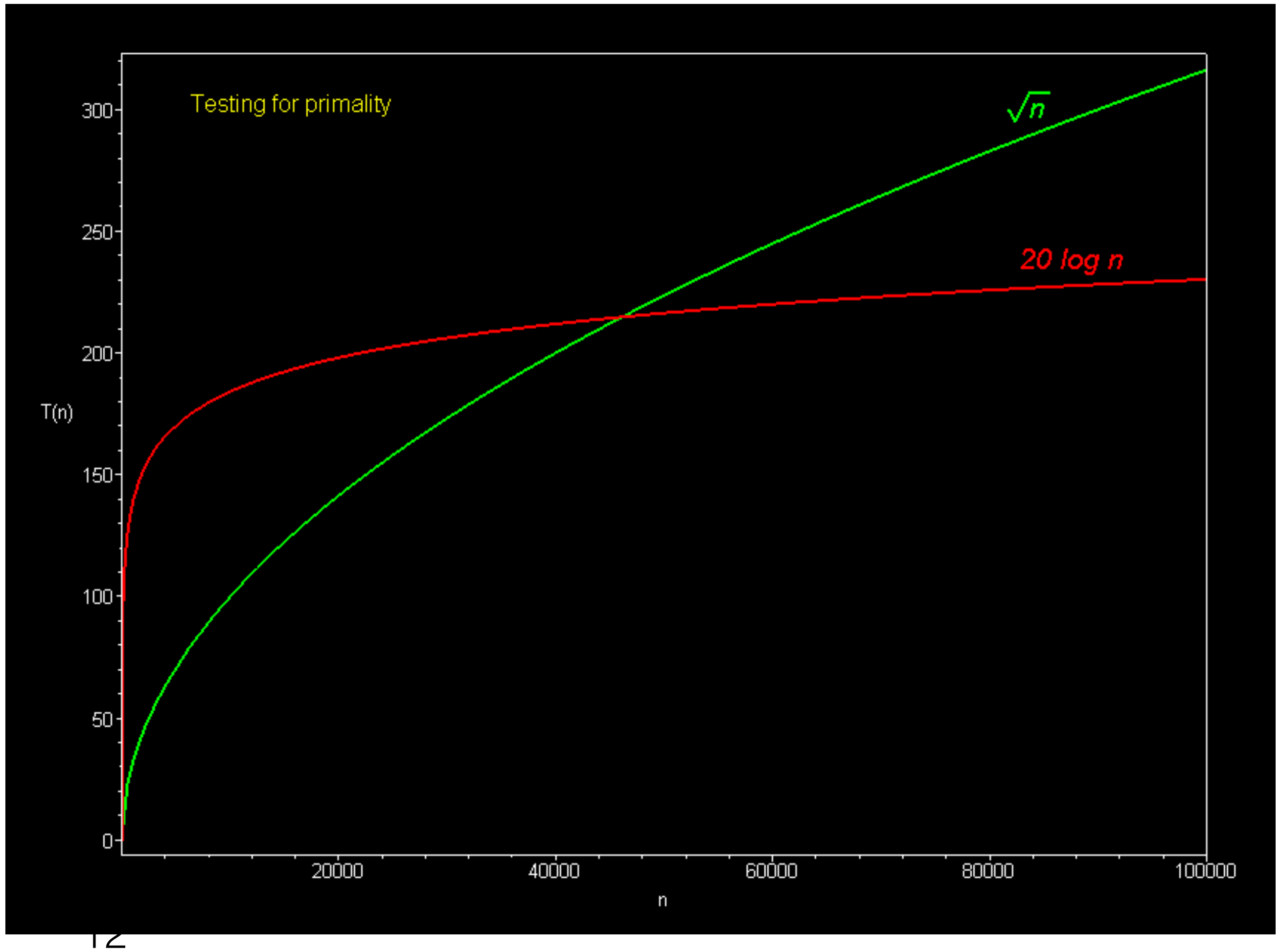


image source: http://science.slc.edu/~jmarshall/courses/2006/fall/accelcs/pub/week15/BigO/

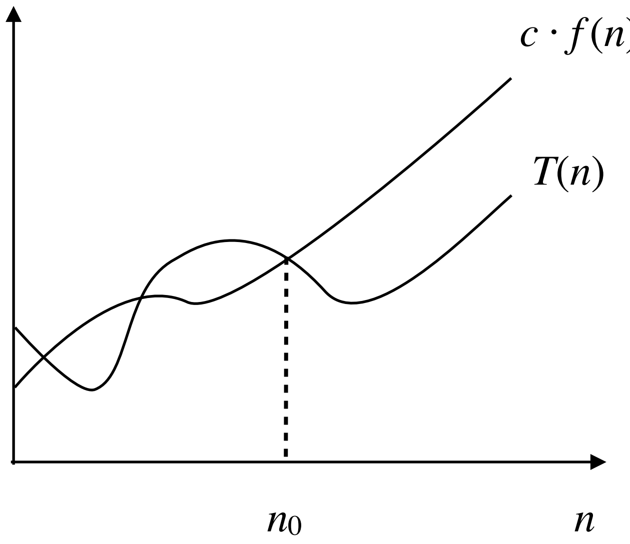
Graphical solution





Big-Oh notation

Combinatoric definition. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.



Analytical definition. $T(n)$ is $O(f(n))$ if

$$\limsup_{n \to \infty} \frac{T(n)}{f(n)} < \infty.$$

Intuition: $c \cdot f(n)$ is an *upper* bound on T(n) on any input. (if n is sufficiently large)

$3n^2 + 2n + 1$ is $O(n^2)$ but also $O(n^3)$

$3n^{1/2} + \log n$ is $O(n^{1/2})$

$n\left(\log n + \sqrt{n}\right)$ is $O(n^{3/2})$

(mock) TopHat question

Which of these are False?

A. $10^6 n^3 + 5n^2 - n + 10 = O(n^3)$

B. $\sqrt{n} + logn = O(\sqrt{n})$

C. $n = O(n^3)$

D. $n(\sqrt{n} + logn) = O(\sqrt{n})$

Notation

- Domain. The domain of $f(n)$ is typically the natural numbers $\{ 0, 1, 2, \ldots \}$
  - we consider exceptions to be implied as needed
  - Ex. $\log_2 n$ is not defined when $n=0$

- Non-negative functions. When using big-Oh notation, we consider asymptotic order based on their absolute value
  - $-10n^2 = O(n^2)$ and not $O(n)$
  - (in case f(n) is representing the running time of an algorithm negative values don't make sense of course)

- Equals sign. $O(f(n))$ is a set of functions, but computer scientists often write $T(n) = O(f(n))$ instead of $T(n) \in O(f(n))$.

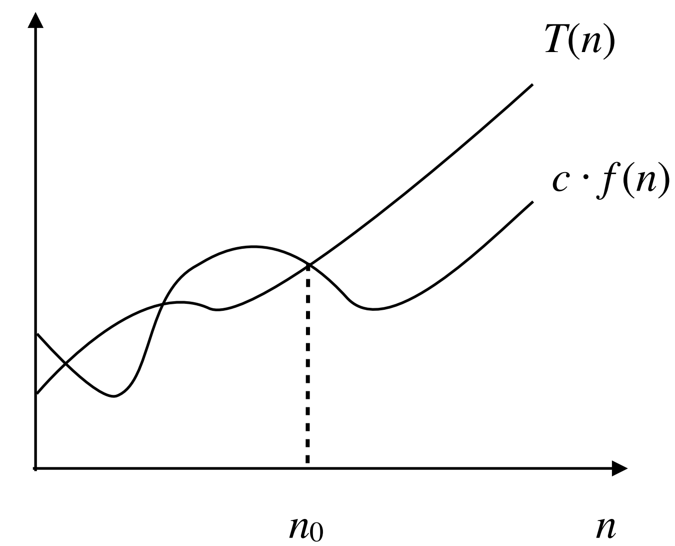
Big-Oh is not symmetric

$f(n) = 5n^3$ and $g(n) = 8n^2$ then

- $f(n) = O(n^3)$
- $g(n) = O(n^3)$
- but $f(n) \mathrel{!=} O(g(n))$

Big-Omega notation

Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$.



Ex. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is both $\Omega(n^2)$ and $\Omega(n)$.
- $T(n)$ is neither $\Omega(n^3)$ nor $\Omega(n^3 \log n)$.

Analytical definition: $T(n)$ is $\Theta(f(n))$

$$limsup_{n\to\infty} \frac{f(n)}{T(n)} < \infty$$

(mock) TopHat question

Which of these are False?

A. $10^6n^3 + 5n^2 - n + 10 = \Omega(n^3)$

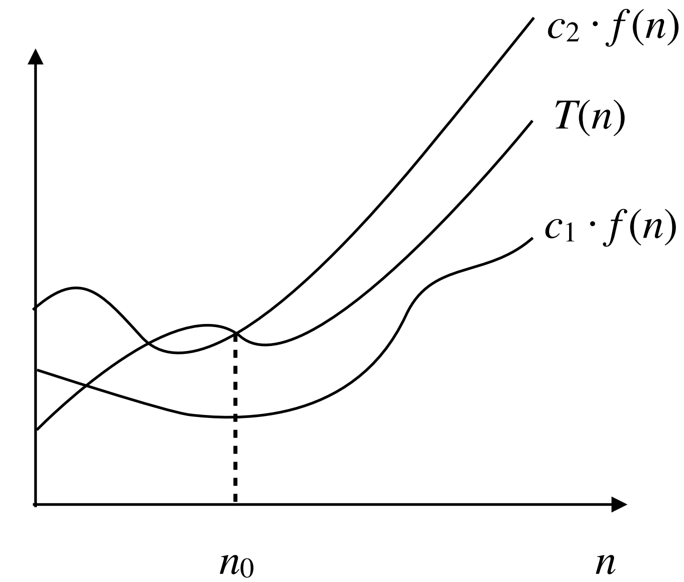
B. $\sqrt{n} + logn = \Omega(\sqrt{n})$

C. $n = \Omega(n^3)$

D. $n(\sqrt{n} + logn) = \Omega(\sqrt{n})$

Big-Theta notation

Tight bounds. $T(n)$ is $\Theta(f(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.



Ex. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is $\Theta(n^2)$. ← choose $c_1 = 32$, $c_2 = 50$, $n_0 = 1$
- $T(n)$ is neither $\Theta(n)$ nor $\Theta(n^3)$.

Analyticial definition. $T(n)$ is $\Theta(f(n))$

$$limsup_{n\to\infty} \frac{f(n)}{T(n)} = \text{const} > 0$$

Typical usage. Mergesort makes $\Theta(n \log n)$ compares to sort n elements.

(mock) TopHat question

Which of these are False?

A. $10^6 n^3 + 5n^2 - n + 10 = \Theta(n^3)$

B. $\sqrt{n} + logn = \Theta(\sqrt{n})$

C. $n = \Theta(n^3)$

D. $n(\sqrt{n} + logn) = \Theta(\sqrt{n})$

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

Pf. $$\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d > 0$$

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

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

Pf. $$\log_a n = \frac{\log_b n}{\log_b a} = \Theta(\log_b n)$$ ← change of base of the logarithm formula

We won't bother with the base of the logarithm when analyzing running times, we will simply use log(n) in the formulas.

- we often assume it's base-2 logarithm because we like binary

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. For every $d > 0$, $\log n$ is $O(n^d)$.

- the logarithm grows slower than any polynomial. (e.g. $n^{1.001}, \sqrt{n}$)

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. For every $d > 0$, $\log n$ is $O(n^d)$.

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

Pf. $$\lim_{n \to \infty} \frac{n^d}{r^n} = 0$$

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. For every $d > 0$, $\log n$ is $O(n^d)$.

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

Factorial. n! grows faster than any polynomial function: $n! = 2^{\Theta(n \log n)}$

- follows from Stirling's formula
- note that $2^n < 2^{\Theta(n \log n)}$

Max vs. Sum

$O(\max\{f(n), g(n)\}) = O(f(n) + g(n))$

Compute $\limsup \dfrac{f(n) + g(n)}{f(n)}$

Big-Oh notation with multiple variables

Upper bounds. $T(m, n)$ is $O(f(m, n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and $n_0 \geq 0$ such that $T(m, n) \leq c \cdot f(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

Ex. $T(m, n) = 32mn^2 + 17mn + 32n^3$.

- $T(m, n)$ is $O(mn^2 + n^3)$. — we don't know which of the two is larger, hence we keep both
- $T(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $\Theta(n + m)$ time to find the shortest path from s to t in a graph. (Here m is the number of edges, n the number of nodes in a graph.)

review

n factorial

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 =$ number of permutations of n numbers

review

n factorial

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1 =$ number of permutations of n numbers

combinations - “n choose k”

$$\binom{n}{k} = \frac{n!}{(n-k)!k!} = \frac{n \cdot (n-1) \ldots \cdot (n-k+1)}{k \cdot (k-1) \ldots \cdot 2 \cdot 1}$$

number of ways to choose k items out of a set of n without repetition when the order doesn’t matter.

Asymptotic bounds for some common functions - for your review

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

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

Logarithms and polynomials. For every $d > 0$, $\log n$ is $O(n^d)$.

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

Factorial. n! grows faster than any polynomial function: $n! = 2^{\Theta(n \log n)}$

- follows from Stirling's formula
- note that $2^n < 2^{\Theta(n \log n)}$
- in the homework you may use $\Theta(n!)$

Examples - compare by Θ

1. $\sqrt{n}$ $2^{\log_5 \sqrt{n}}$

2. n^{10^6} $1.000001^{\sqrt{n}}$ $n^{\frac{n}{2}}$

Max vs. Sum

$$O(\max\{f(n), g(n)\}) = O(f(n) + g(n))$$

Compute $\limsup \frac{f(n) + g(n)}{f(n)}$

Big-Oh notation with multiple variables

Upper bounds. $T(m, n)$ is $O(f(m, n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and $n_0 \geq 0$ such that $T(m, n) \leq c \cdot f(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

Ex. $T(m, n) = 32mn^2 + 17mn + 32n^3$.

- $T(m, n)$ is $O(mn^2 + n^3)$. — we don't know which of the two is larger, hence we keep both
- $T(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $\Theta(n + m)$ time to find the shortest path from s to t in a graph. (Here m is the number of edges, n the number of nodes in a graph.)

Graphs

A graph G(V,E) consist of a pair

set of vertices (nodes) V

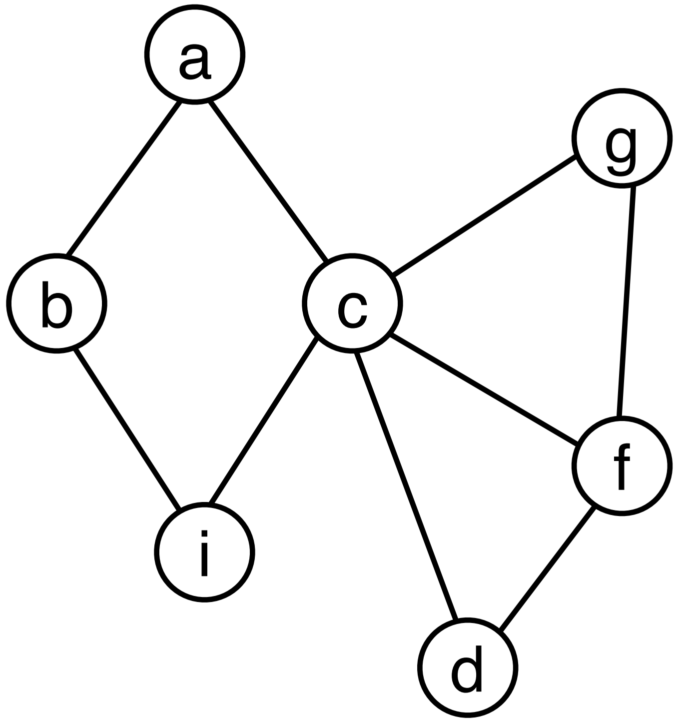
and a set of edges E; an edge e = (u,v) is a pair of vertices

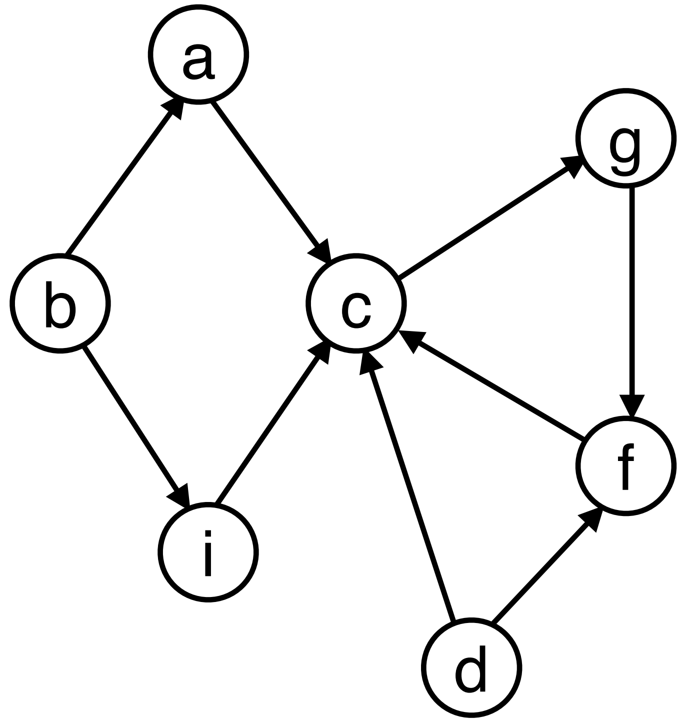
G is undirected if the edges (u,v) are *unordered* pairs.

- (u,v) and (v,u) have the same meaning; the two are connected

G is directed if the edges (u,v) are *ordered* pairs.

- we say there is an edge from u to v

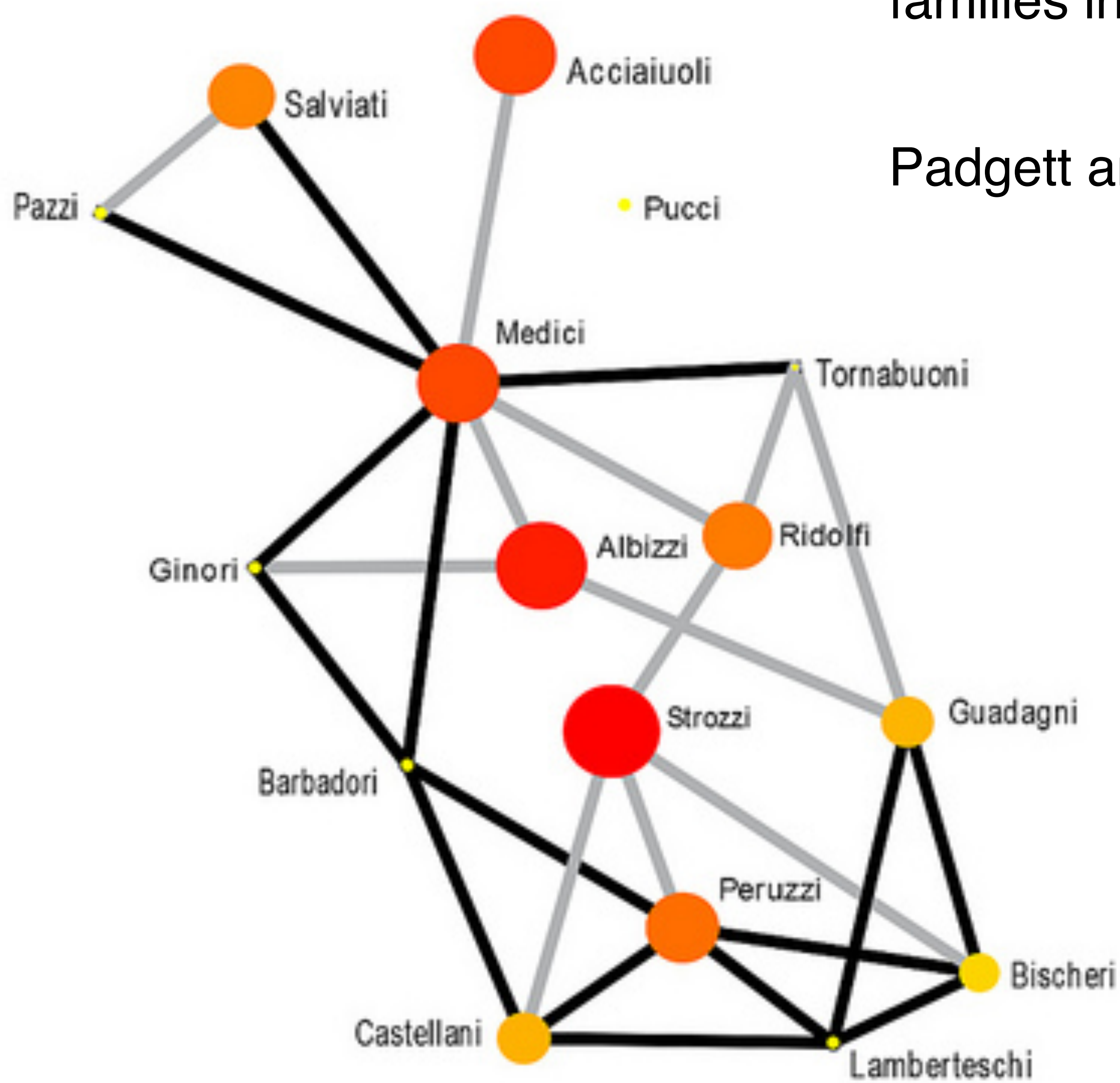


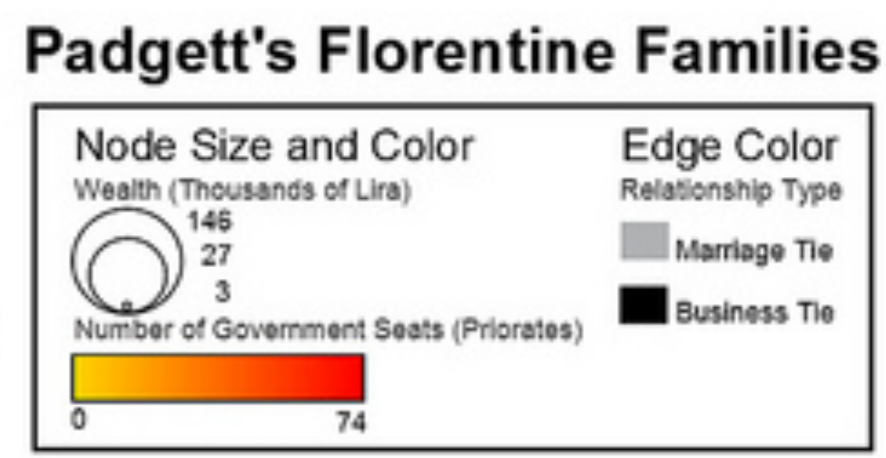


Florentine Families

Marriage and business ties between families in 15th century Florence.

Padgett and Ansell 1993





Graphs

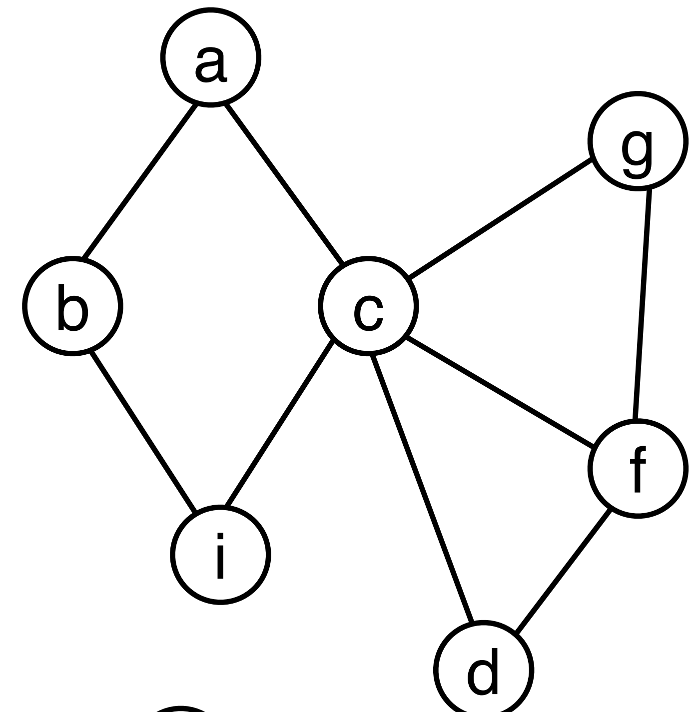
A graph G(V,E) consist of a pair

set of vertices (nodes) V — number of vertices |V|

and a set of edges E; an edge e = (u,v) is a pair of vertices — number of edges |E|

G is undirected if the edges (u,v) are *unordered* pairs.

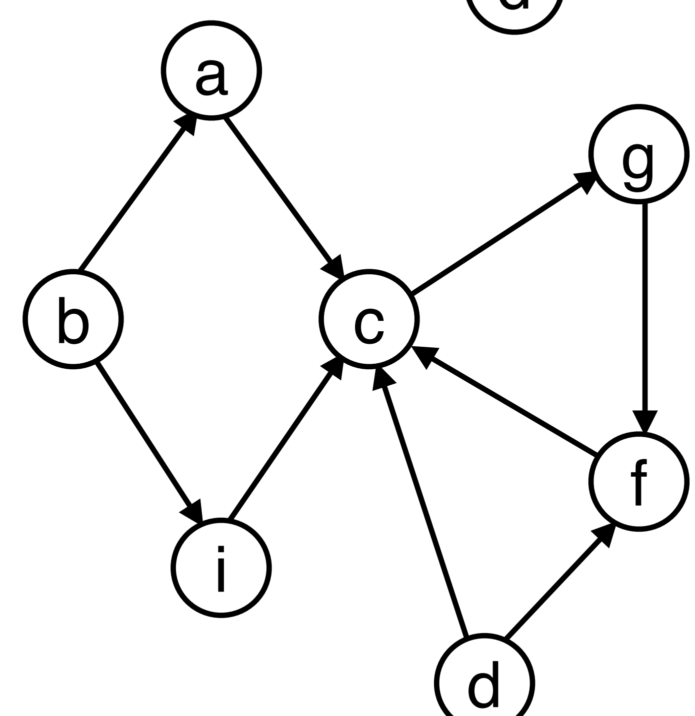
- degree(v) = number of edges adjacent to v



G is directed if the edges (u,v) are *ordered* pairs.

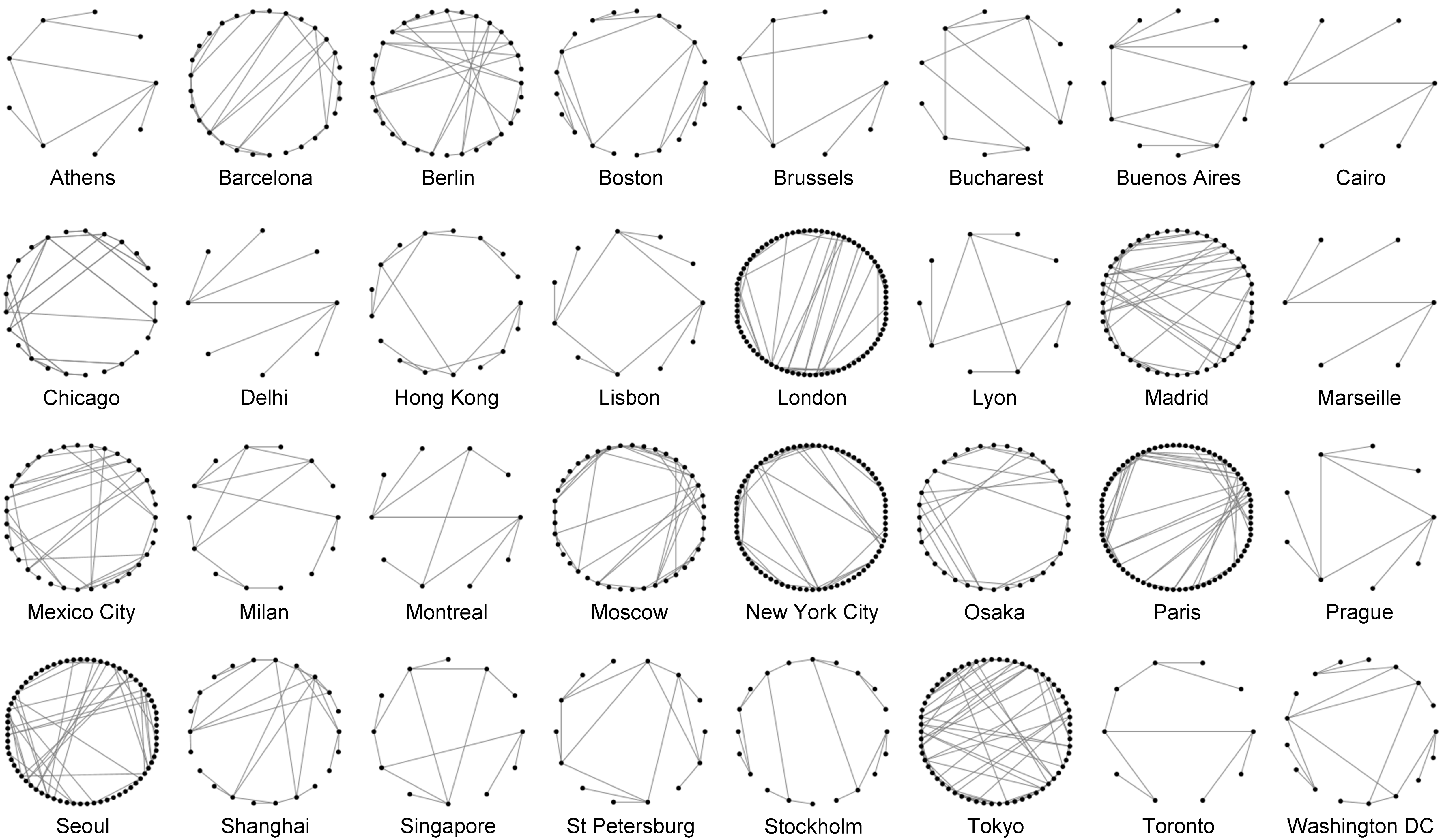
- outdegree(v) = number of edges directed from v
- indegree(v) = number of edges directed to v

- notation: often we use |V| = n, |E| = m



MBTA subway map

Graphs of subway networks in major cities

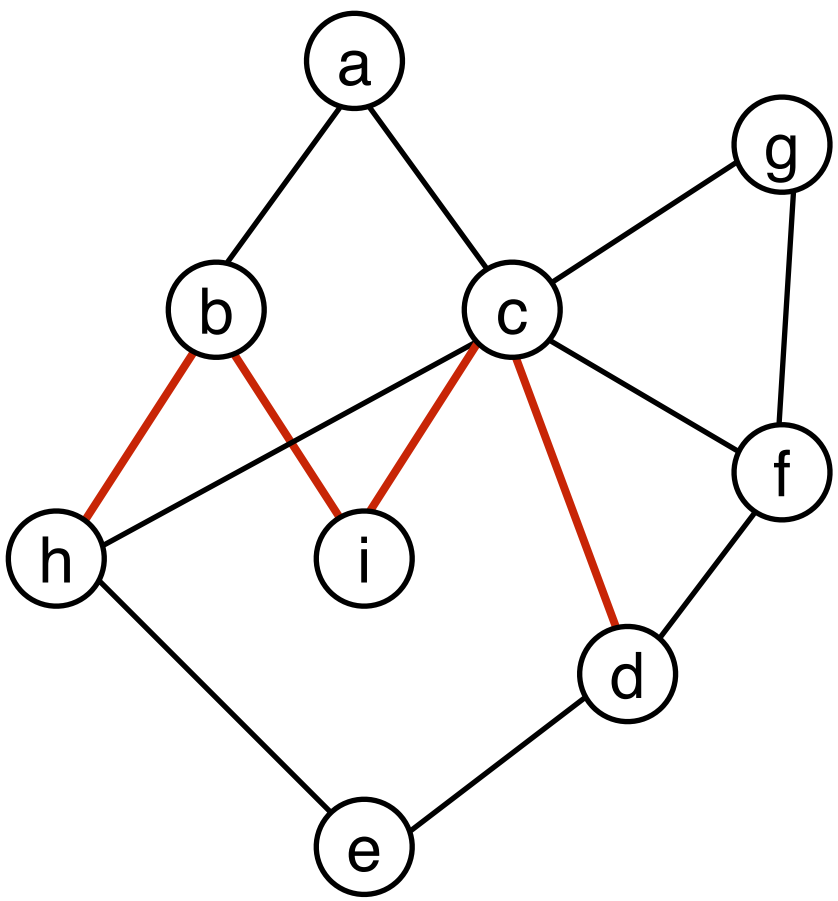


S. Derrible, Network Centrality of Metro Systems, PLOS One, 2012

Paths and connectivity

Def. A path in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair v_{i-1}, v_i is joined by an edge in E.

Def. A path is simple if all nodes are distinct.

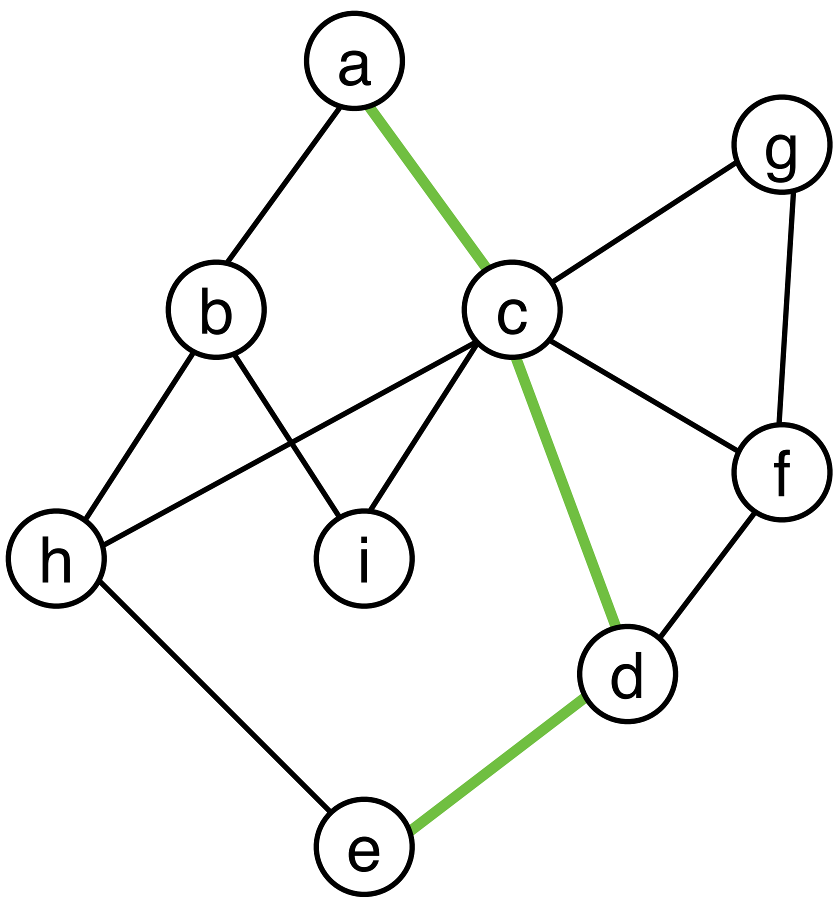


path: h - b - i - c - d

Paths and connectivity

Question. Is there a path from a to e? *yes: a-c-d-e*

Def. An undirected graph is connected, if there is a path between any pair of nodes.



Can I get from Boston to NYC by car?

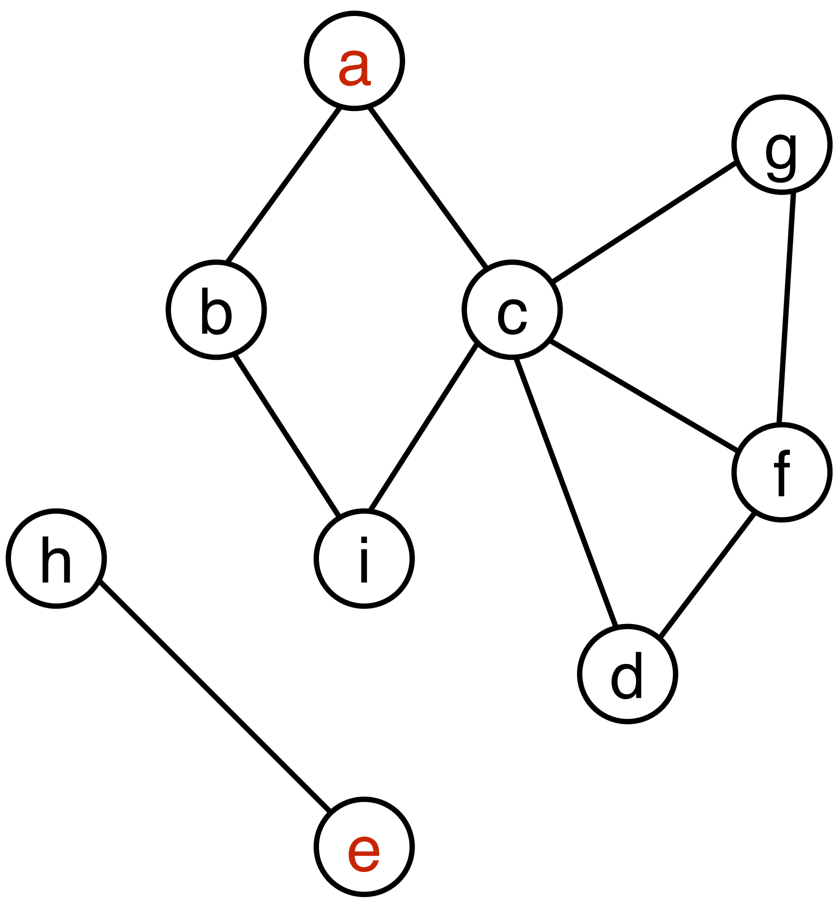
From Boston to London UK?

Paths and connectivity

Question. Is there a path from s to e?

Def. An undirected graph is connected, if there is a path between any pair of nodes.

Task: Given a source node s, find all nodes connected to s.



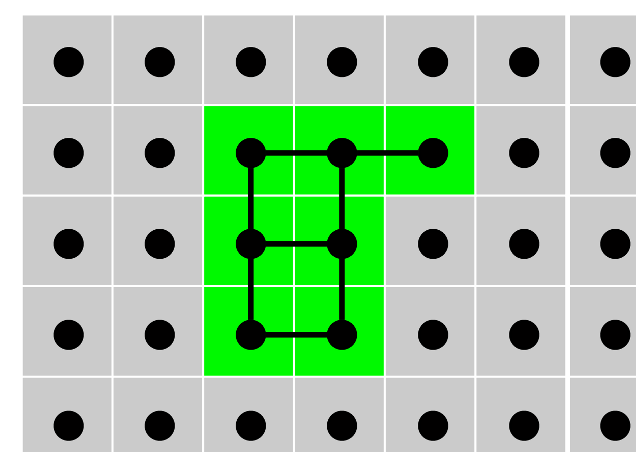
Min and max number of edges in a connected graph?

Application of connectivity: Flood fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

recolor lime green blob to blue



Application of connectivity: Flood fill

Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.

- Node: pixel.
- Edge: two neighboring lime pixels.
- Blob: connected component of lime pixels.

recolor lime green blob to blue