Stable Matching & Asymptotic Analysis

Reminders

- Make sure you are on the **course slack** and have filled out the course **introduction google form**
- If you have not done so already: sign up for introductory meetings
    - If no time on it works, show up to office hours
- Register on Gradescope using course code **74XDKB**
- Assignment 0 due **Wed, Fab 24 at 11 pm**
- Zoom help hours:
    - (Today) Me: 2.30-4 pm, TAs: 3.30-5.30 pm, 7-9 pm
    - (Tomorrow) Me: 3-5 pm, TAs: 8-10 pm

Stable Matching Problem

Input. A set H of n hospitals, a set S of n students and their preferences

Goal. Find a perfect matching M s.t. there are no unstable pairs, that is, there does not a pair $(h, s) \in H \times S$

- h prefers s to its current match in M, and
- s prefers h to its current match in M

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Stable Matching Problem

Input. A set H of n hospitals, a set S of n students and their preferences

Goal. Find a perfect matching M s.t. there are no unstable pairs, that is, there does not a pair $(h, s) \in H \times S$

- h prefers s to its current match in M, and
- s prefers h to its current match in M

False start.

- Each hospital makes offer to its top available candidate
- Each student accepts its top offer and rejects others

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris

	1st	2nd	3rd
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	2nd	3rd
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH

	1st	2nd	3rd
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	2nd	3rd
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

False Starts

Proceed greedily in rounds until matched.

- (Round 1) MA → Aamir, NH → Aamir, OH → Chris
- (Round 1) Aamir rejects MA, accepts NH, Chris accepts OH
- (Round 2) Only Beth and MA left, and must match

Is this a stable matching?

- Unstable pair: (MA, Chris). What could have avoided it?

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched. In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched. In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	2nd	3rd
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	2nd	3rd
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched. In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Deferred Acceptance Algorithm

Proceed in rounds until all hospitals matched. In each round,

- Each free hospital offers to its top choice among candidates it hasn't offered yet
- Each free student *retains but defers accepting* **top offer**, rejects others
- If a student receives a better offer than currently retained, they reject current and retain new offer (trade up)

	1st	**2nd**	**3rd**
MA	Aamir	Chris	Beth
NH	Aamir	Beth	Chris
OH	Chris	Beth	Aamir

	1st	**2nd**	**3rd**
Aamir	OH	NH	MA
Beth	MA	OH	NH
Chris	MA	NH	OH

Gale-Shapely Algorithm

GALE–SHAPLEY (*preference lists for hospitals and students*)

INITIALIZE M to empty matching.

WHILE (some hospital h is unmatched and hasn't proposed to every student)

  $s \leftarrow$ first student on h's list to whom h has not yet proposed.

  IF (s is unmatched)

    Add h–s to matching M.

  ELSE IF (s prefers h to current partner h')

    Replace h'–s with h–s in matching M.

  ELSE

    s rejects h.

RETURN stable matching M.

Analyzing Gale-Shapely

Questions to ask

Correctness:

- Does it match everyone? (produce a perfect matching)
- Does it produce a stable matching?

Efficiency:

- How long does it take to produce a matching?
- We will review Big Oh before we analyze this

Analyzing the Algorithm: Correctness

Does it match everyone? (Perfect matching)

- Once a student gets an offer: has at least a tentative match
- Equivalently, if a student is unmatched, then no hospital has offered them
    - Some hospital has not exhausted its preference lists
- When the algorithm terminates, everyone is matched (i.e., it produces a *perfect matching)*.

Does it produce a stable matching?

- Key idea: students always 'trade up'
- s breaks match with h in favor of h' only if s prefers h' to h

Analyzing the Algorithm: Correctness

Lemma. The Gale Shapely Algorithm produces a stable matching.

Proof. (By contradiction) Let M be the resulting matching. Suppose $\exists (h, s)$ such that $(h, s'), (h', s) \in M$ and

- h prefers s over s' and s prefers h over h'

Notice that h must have offered to s before s'

- Either s broke the match to h at some point, or s already had a match h'' that s preferred over h

But students always trade up, so s must prefer final match h' over h'', which they prefer over h. $(\Rightarrow\Leftarrow)$ ∎

Analyzing Gale-Shapely

Questions to ask

Correctness:

- Does it match everyone? (produce a perfect matching)
- Does it produce a stable matching?

Efficiency:

- How long does it take to produce a matching?
- How do we usually we measure this?

Measuring Complexity

- What is a good measure of performance of an algorithm?
- What constitutes an efficient algorithm?
  - Runs quickly on large, ‘real’ instances of problems
  - Qualitatively better than brute force
  - Scales well to large instances

Brute Force: Often Inefficient

- Efficient: Qualitatively better than brute force
- Brute force: often exponentially large because
  - Might examine all subsets of a set: 2^n
  - Might examine all orderings of a list: $n!$
- But 2^n is still not efficient even though it's qualitatively better than $n!$

Measuring Complexity : Scalability

- Desirable scalability property. When the input size doubles the algorithm should slow down by at most some constant factor C
- Examples
  - $f(n) = n^k$, then $f(2n) = 2^k n^k = cn^k$ for any fixed k
  - $g(n) = \log n$, then $g(2n) = \log 2 + \log n \leq c \log n$ for $n \geq 2$
- But not for these functions
  - $f(n) = 2^n$, then $f(2n) = 2^{2n} = (2^n)^2$
  - $g(n) = n!$, then $g(2n) = 2n! \geq n^n \cdot n!$
- An algorithm is polynomial time if the above scaling property holds, i.e., its running time is bounded above by a polynomial function of the input size n

Growth of Functions

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	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

Worst Case Analysis

- But how do we measure running time?
- **Worst-case running time:** the maximum number of **steps** needed to solve a *problem instance* of size n
- Overestimates the typical runtime but gives strong guaranties
- “I promise you that my algorithm is ALWAYS this fast!”
- Often there’s no easy to identify “worst” case
  - Don’t fall into the “the worst case is when…” trap!

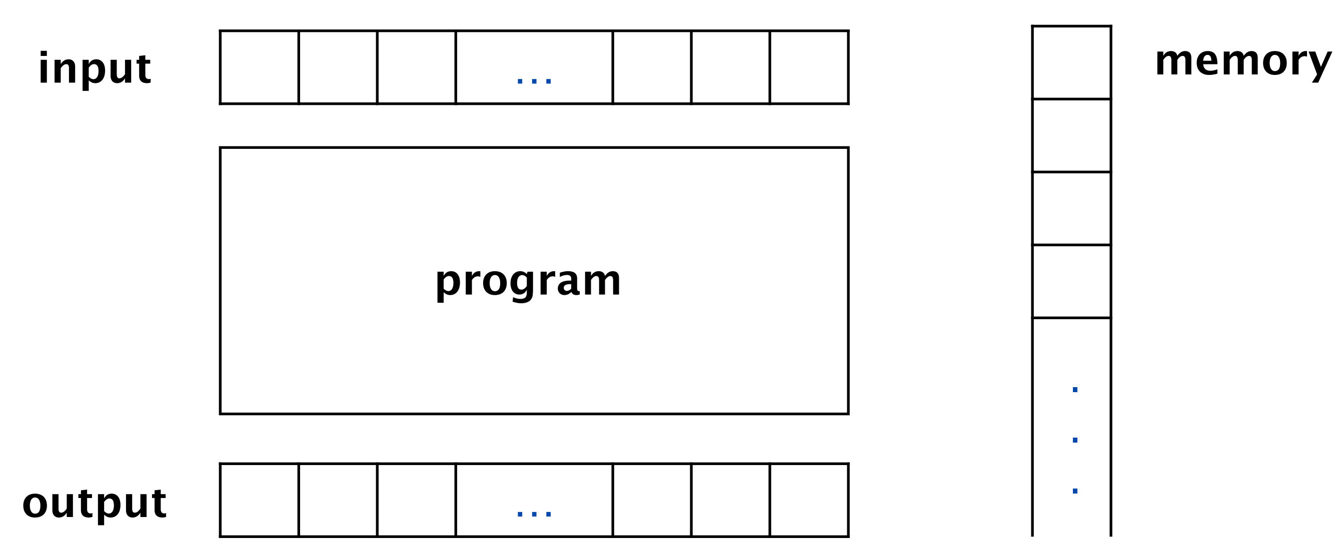
Other Types Of Analysis

- **Probabilistic.** Expected running time of a randomized algorithm
  - e.g., the expected running time of quicksort

- **Amortized.** Worst-case running time for any sequence of n operations
  - Some operations can be expensive but may make future operations fast (doing well on average)
  - e.g., Union-find data structure (we'll study in a few weeks)
- Average-case analysis, smoothed analysis, competitive analysis, etc.

How to Measure Cost

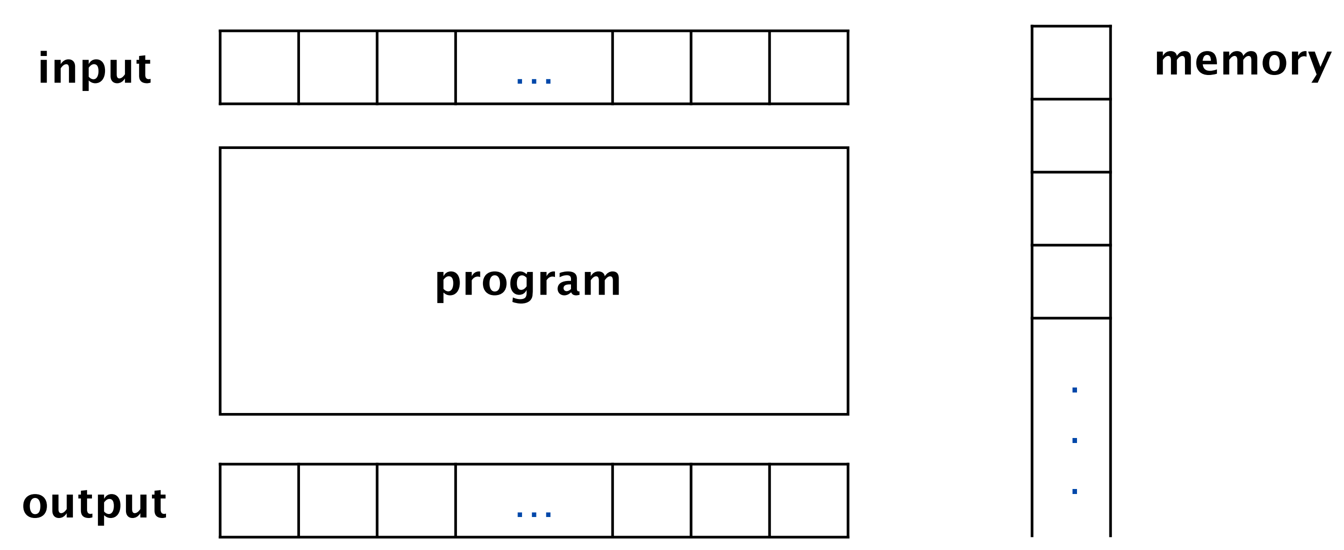
- “Word RAM” model of computation
- **Basic idea**: every operation on a primitive type in C, Java, etc. costs 1 unit of time:
  - Adding/multiplying/dividing/etc two ints or floats costs 1
  - An if statement costs 1
  - A comparison costs 1
  - Dereferencing a pointer costs 1
  - Array access costs 1



Model of Computation Details

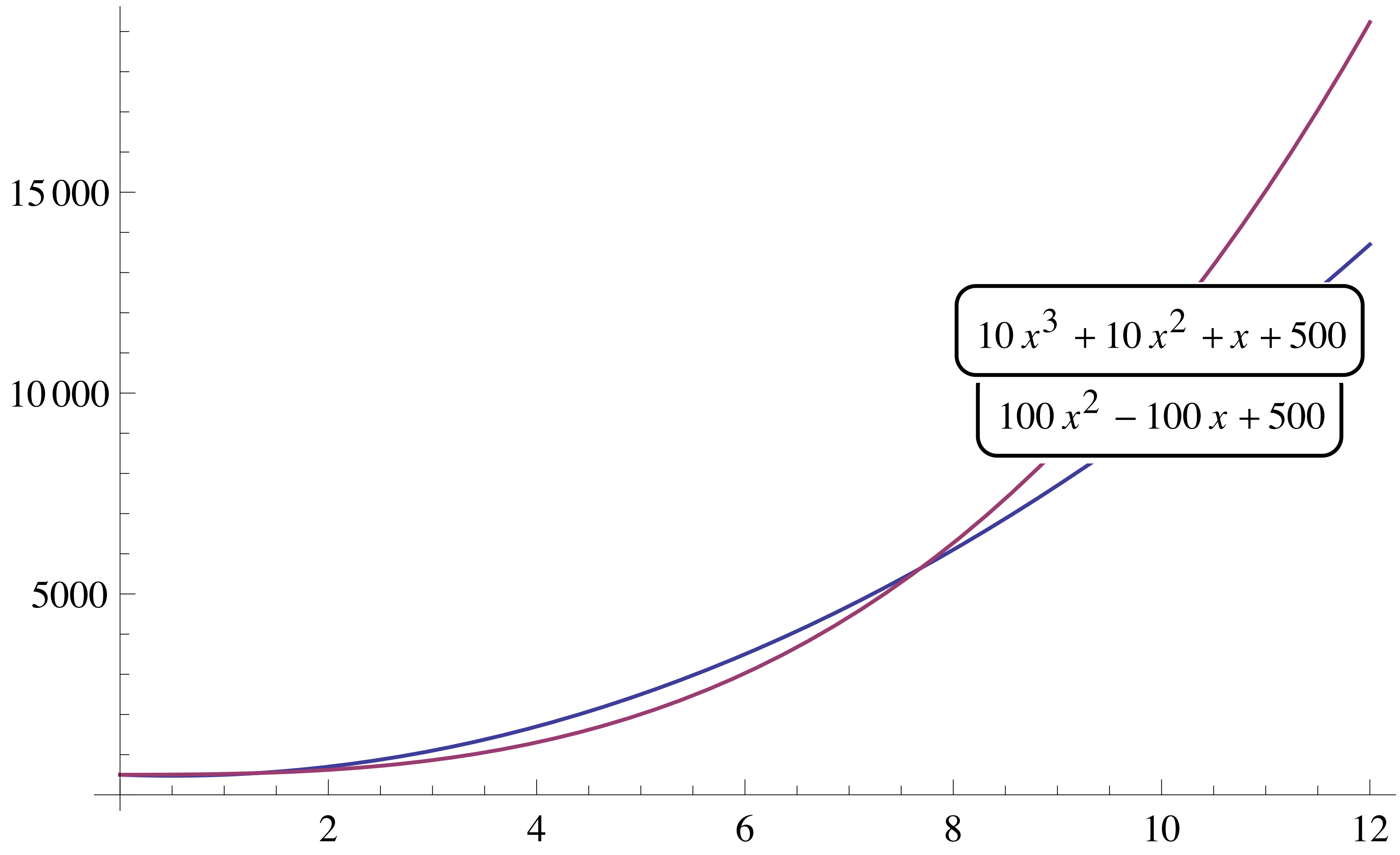
- **Word RAM model**
  - Each memory location and input/output cell stores a w-bit integer (assume $w \geq \log_2 n$)
  - Primitive operations: arithmetic operations, read/write memory, array indexing, following a pointer etc. are constant time
- **Running time:** number of primitive operations
- **Space:** number of memory cells utilized

Space is measured in "words" (ints, floats, chars, etc) not bits



Asymptotic Growth

What matters: How functions behave “as n gets large”



Asymptotic Upper Bounds

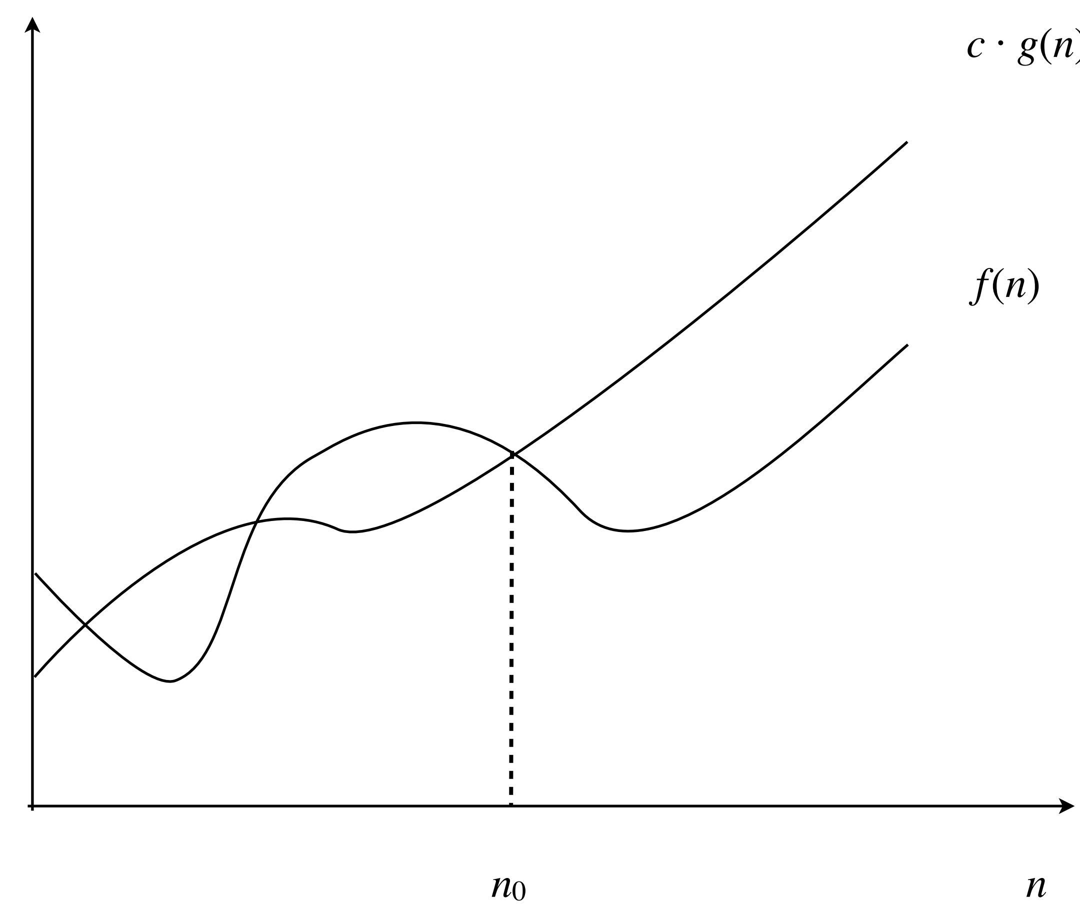
Definition: $f(n)$ is $O(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

In other words, for sufficiently large n, $f(n)$ is asymptotically bounded above by $g(n)$

Examples

- $100n^2 = O(n^2)$
- $n \log n = O(n^2)$
- $5n^3 + 2n + 1 = O(n^3)$

Typical usage. Insertion sort makes $O(n^2)$ compares to sort n elements



Class Quiz

Let $f(n) = 3n^2 + 17n\log_2 n + 1000$. Which of the following are true?

A. $f(n)$ is $O(n^2)$.

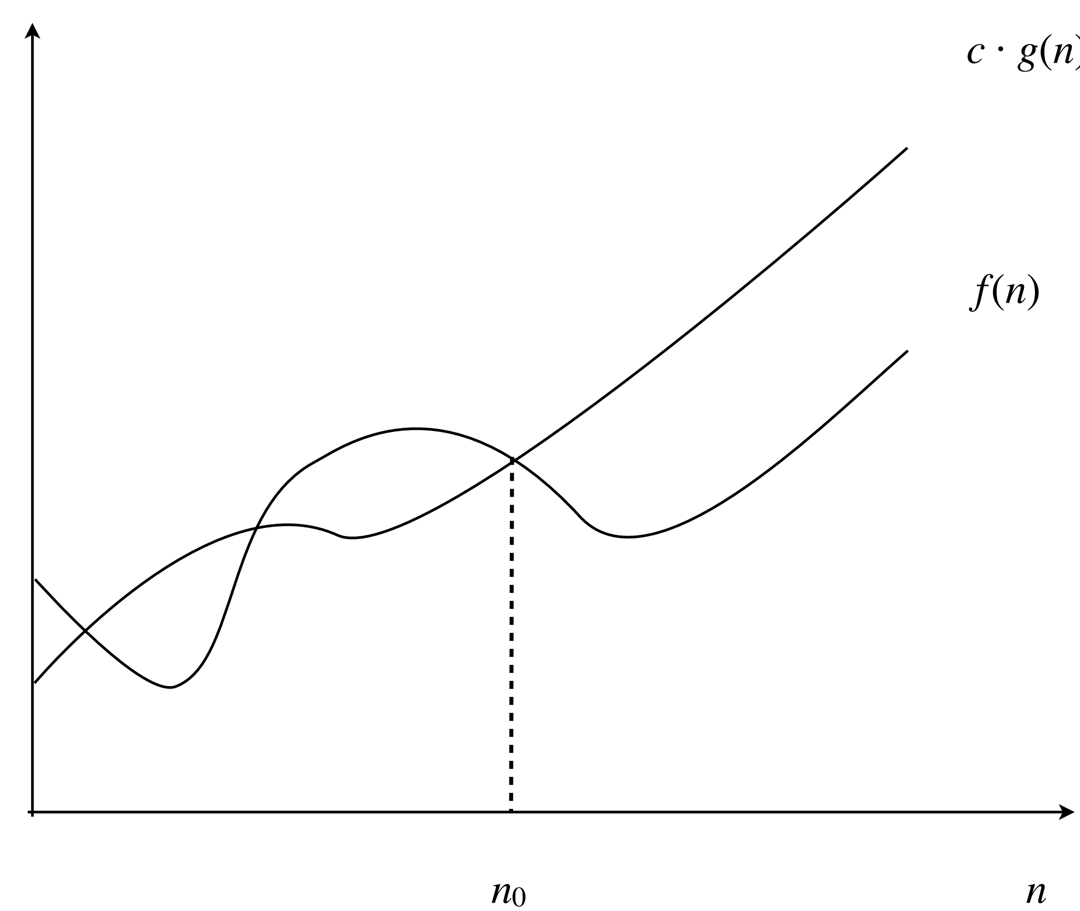
B. $f(n)$ is $O(n^3)$.

C. Both A and B.

D. Neither A nor B.

Big Oh- Notational Abuses

- $O(g(n))$ is actually a set of functions, but the CS community writes $f(n) = O(g(n))$ instead of $f(n) \in O(g(n))$
- For example
  - $f_1(n) = O(n \log n) = O(n^2)$
  - $f_2(n) = O(3n^2 + n) = O(n^2)$
  - But $f_1(n) \neq f_2(n)$
- Okay to abuse notation in this way



Playing with Logs: Properties

- In this class, $\log n$ means $\log_2 n$, $\ln n = \log_e n$
- Constant base doesn't matter: $\log_b(n) = \dfrac{\log n}{\log b} = O(\log n)$
- Properties of logs:
  - $\log(n^m) = m \log n$
  - $\log(ab) = \log a + \log b$
  - $\log(a/b) = \log a - \log b$

$$a^{\log_a n} = n$$

We will use this a lot!

Exponents

$$n^a \cdot n^b = n^{a+b}$$

$$(n^a)^b = n^{ab}$$

Comparing Running Times

- When comparing two functions, helpful to simplify first
- Is $n^{1/\log n} = O(1)$?
- Is $\log \sqrt{4^n} = O(n^2)$?
- Is $n = O(2^{\log_4 n})$?

Comparing Running Times

- When comparing two functions, helpful to simplify first
- Is $n^{1/\log n} = O(1)$?
  - Simplify $n^{1/\log n} = (2^{\log n})^{1/\log n} = 2$: **True**
- Is $\log \sqrt{4^n} = O(n^2)$
  - Simplify $\log \sqrt{2^{2n}} = \log 2^n = n \log 2 = O(n)$: **True**
- Is $n = O(2^{\log_4 n})$?
  - Simplify $2^{\log_4 n} = 2^{\frac{\log_2 n}{log_2 4}} = 2^{(\log_2 n)/2} = 2^{\log_2 \sqrt{n}} = \sqrt{n}$: **False**

Something Missing

- Big Oh is like $\leq$
- So one can accurately say “merge sort requires $O(2^n)$ time,” but it’s not very meaningful
- Can we get terminology like big-O that lower bounds?
- Or that shows two functions are “equal” (up to constants and for large values of n)?

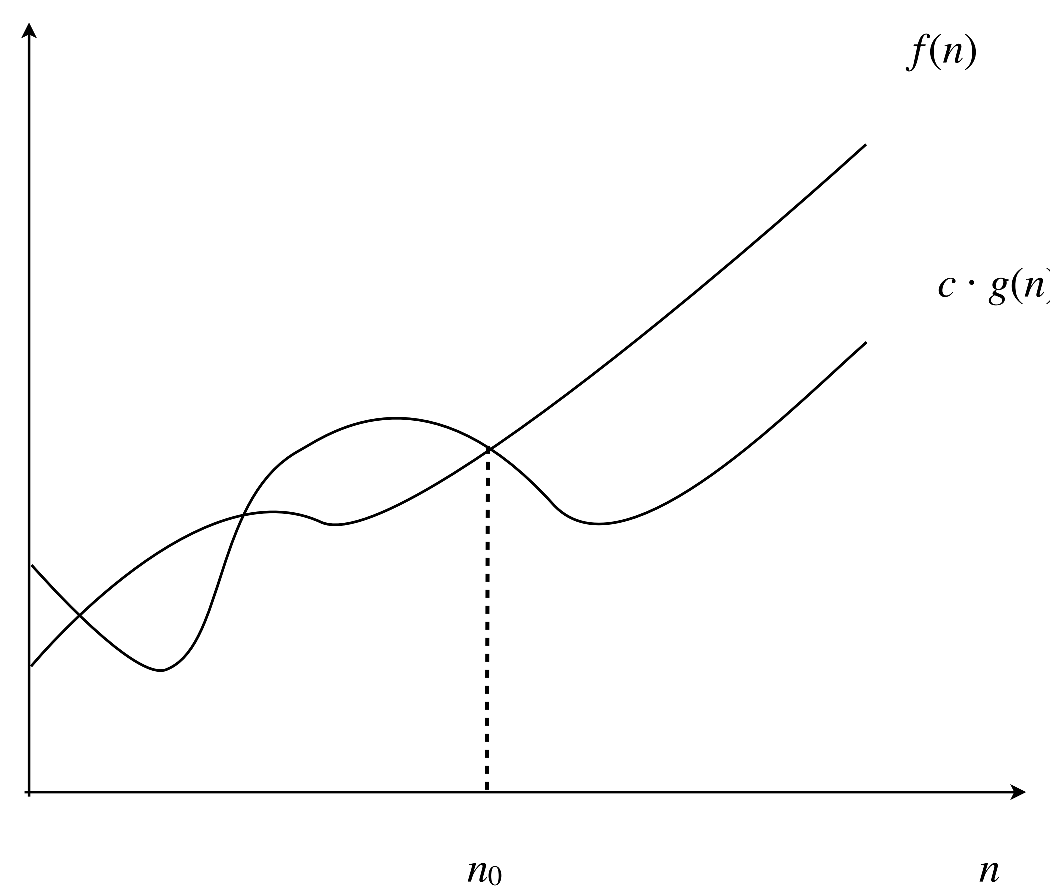
Asymptotic Lower Bounds

Definition: $f(n)$ is $\Omega(g(n))$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all $n \geq n_0$

In other words, for sufficiently large n, $f(n)$ is asymptotically bounded below by $g(n)$. (Same abuse of notation as big Oh)

Examples

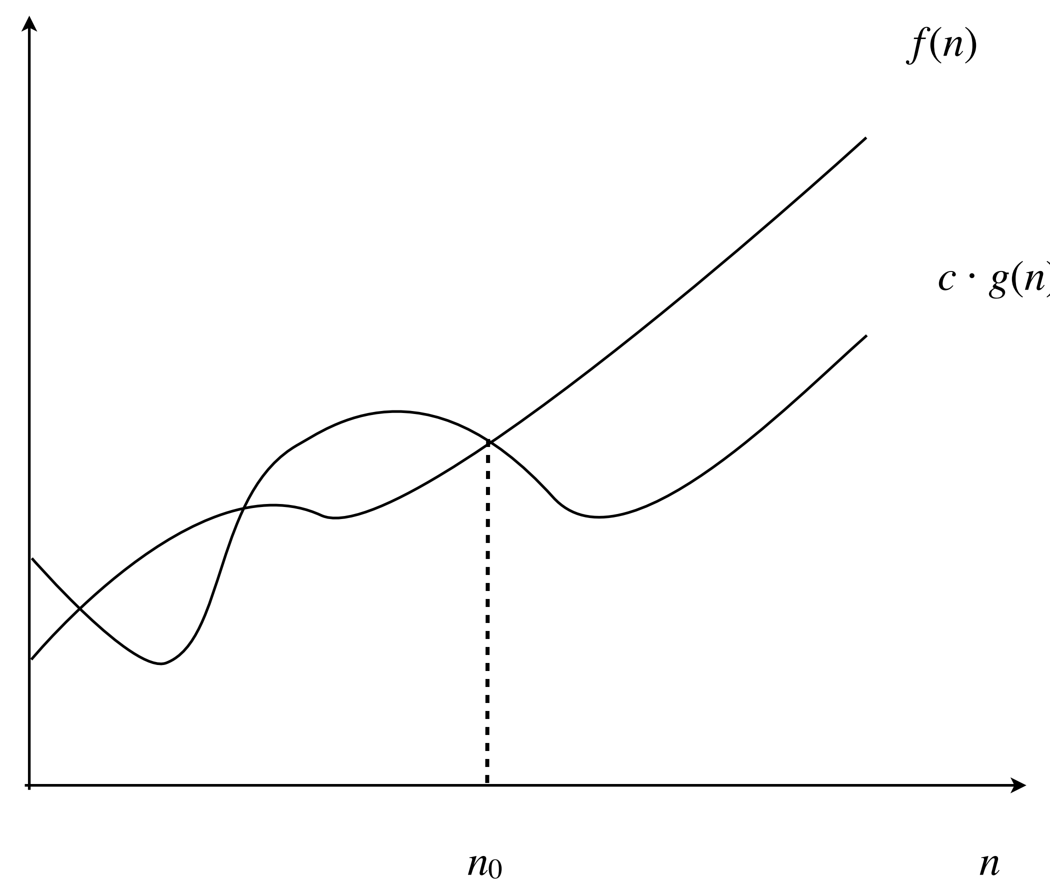
- $100n^2 = \Omega(n^2) = \Omega(n)$
- $n \log n = \Omega(n)$
- $8^{\log n} = \Omega(n^2)$



Why Lower Bounds?

Show that an algorithm performs *at least* so many steps

- Searching an unordered list of n items: $\Omega(n)$ steps in some cases
- Quicksort (and selection/insertion/bubble sorts) take $\Omega(n^2)$ steps in some cases
- Mergesort takes $\Omega(n \log n)$ steps in all cases



Class Quiz

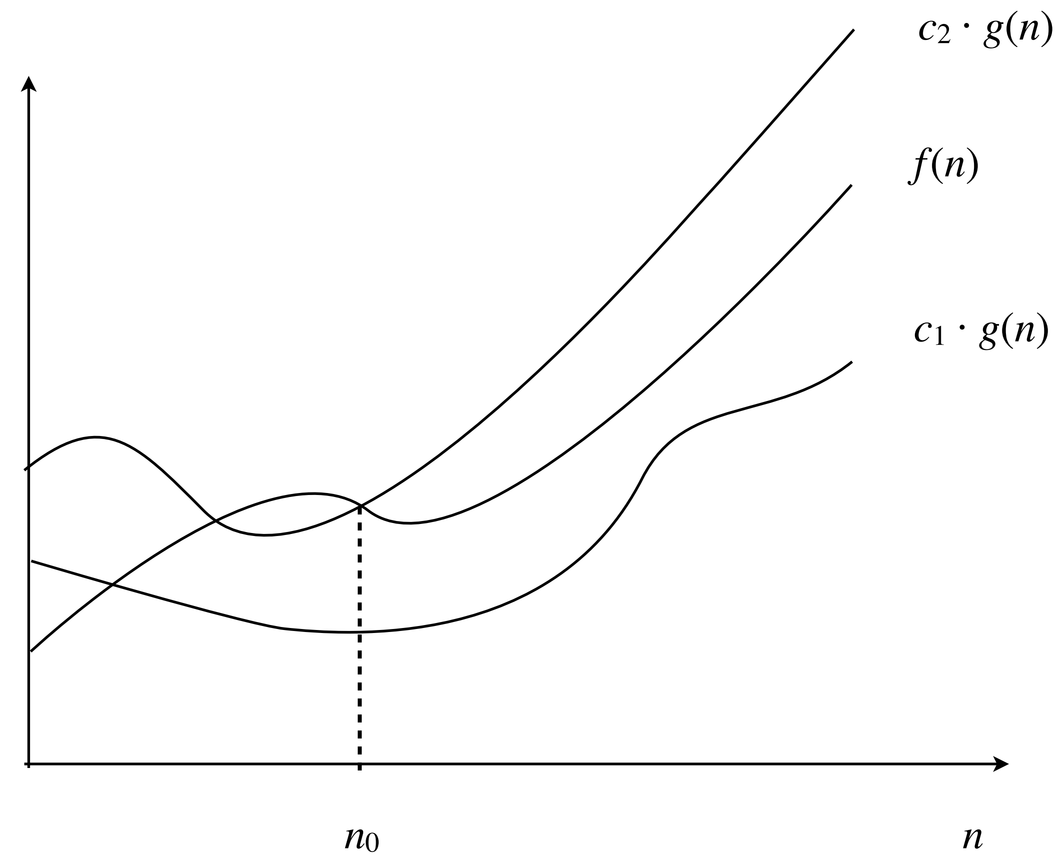
True or False:

$f(n)$ is $\Omega(g(n))$ if and only if $g(n)$ is $O(f(n))$

Asymptotically Tight Bounds

Definition. $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Equivalently, if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.



Examples

- $5n^3 + 2n + 1 = \Theta(n^3)$
- $\log_{100} n = \Theta(\log_2 n)$

Tools for Comparing Asymptotics

- Logs grow slowly than any polynomial:
  - $\log_a n = O(n^b)$ for every $a > 1,\ b > 0$
- Exponentials grow faster than any polynomial:
  - $n^d = O(r^n)$ for every $d > 1,\ r > 0$
- Taking logs
  - As $\log x$ is a strictly increasing function for $x > 0$, $\log(f(n)) < \log(g(n))$ implies $f(n) < g(n)$
  - E.g. Compare $3^{\log n}$ vs 2^n
    - Taking log of both, $\log n \log 3$ vs n
  - Beware: when comparing logs, constants matter!

Tools for Comparing Asymptotics

- Using limits
  - If $\lim_{n\to\infty} \frac{f(x)}{g(x)} = 0$, then $f(x) = O(g(x))$
  - If $\lim_{n\to\infty} \frac{f(x)}{g(x)} = c$ for some constant $0 < c < \infty$, then $f(x) = \Theta(g(x))$

Next Time: Analyzing Gale Shapley

Acknowledgements

- Slides adapted from Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
- Some material taken from Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)