

CSCI 256 Lecture 3:

Asymptotic Analysis

Shikha Singh

Announcements

- Hand in Exercise #2, Pick up Exercise #3
- Reminder: Assignment 0 due tonight at 10 pm
- **Help hours today:**
 - 1.30 - 3 pm (my office TCL 304)
 - 7-9.30 TA hours in TCL 206
- Updated Tuesday office hours: 11 am - 1 pm from next week
- Assignment 1 will be posted later today
 - Make sure to get in touch with your partners!

Last Time: Analyzing Gale-Shapely

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

Today: Analyzing Gale-Shapely

Correctness:

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

Efficiency:

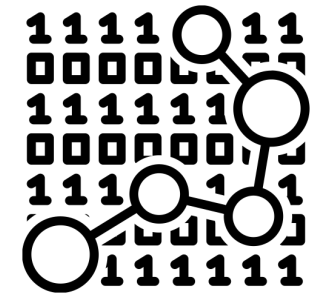
- How long does it take to produce a matching?
- We will review Big Oh before we analyze this

Algorithm Analysis Model

- "Running time" of algorithms measured as **number of steps**
- Every **primitive operation** is 1 step (takes 1 unit of time):
 - Adding/multiplying/dividing/etc two ints
 - An if statement
 - A comparison
 - Array access/update
 - Following a pointer
 - Each iteration of a loop, etc.

Algorithm Analysis Model

- Running time of algorithms measured as number of steps
- Number of step as a function of **input size**
(typically written as n)
- Care about scalability as n gets larger



$$|x| = n$$

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.

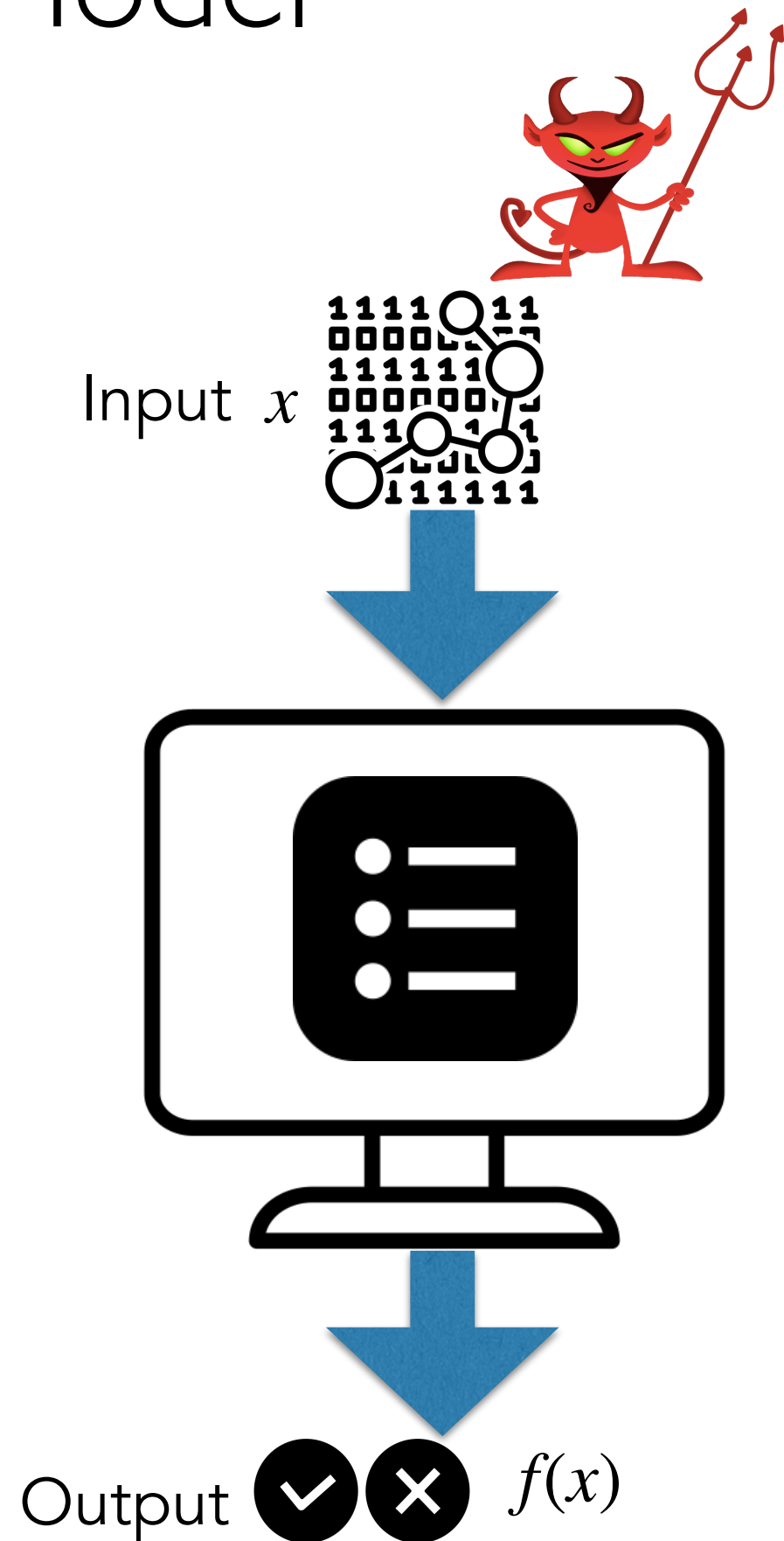
	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	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

Polynomials are Scalable

- **Scalability.** 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 $g(2n) = \log 2 + \log n \approx \log n$
- But not for these functions
 - $f(n) = 2^n$, then $f(2n) = 2^{2n} = (2^n)^2$
 - $g(n) = n!$, then $g(2n) = 2n! \geq n^n \cdot n!$

Algorithm Analysis Model

- Number of steps depend on the input
- For any algorithm, some inputs are easy, others are hard
- **Worst-case paradigm:** Performance measured as the *maximum number of steps* taken on *any* input of size n
 - "Adversarial" analysis



Why Give Worst Case Guarantees?

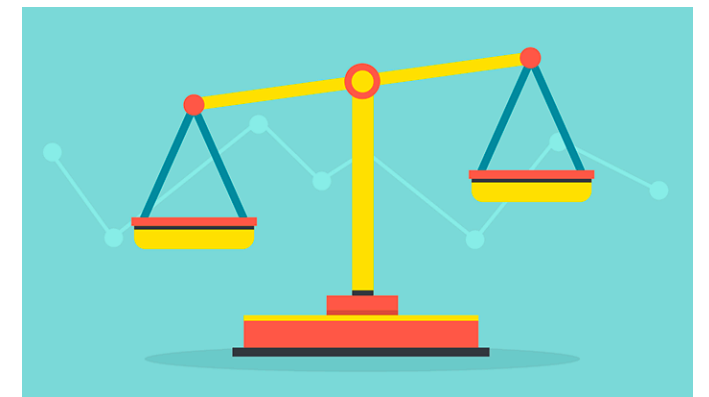
- Worst-case analysis: dominant algorithm design paradigm



Powerful guarantee



Universal



Mathematically
compare algorithms

- **Downside.** Can be too pessimistic on typical everyday (non-worst-case) instances (on which the algorithm might be faster)

[Aside] Beyond-Worst-Case Analyses

- **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)
- **[New] Algorithms with predictions.**
 - Obtain a prediction about the input instance or solution
 - Running time proportional to error in predicted instances



Model of Computation Details

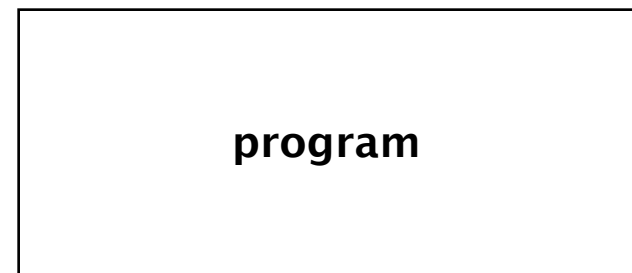
- Word RAM model
 - Each memory location and input/output cell stores a "**word**" which is a w -bit integer (assume $w \geq \log_2 n$)
 - **Primitive operations:** arithmetic operations, read/write memory, array indexing, following a pointer etc. are constant time
- **Running time:** number of primitive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

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

input



program



output

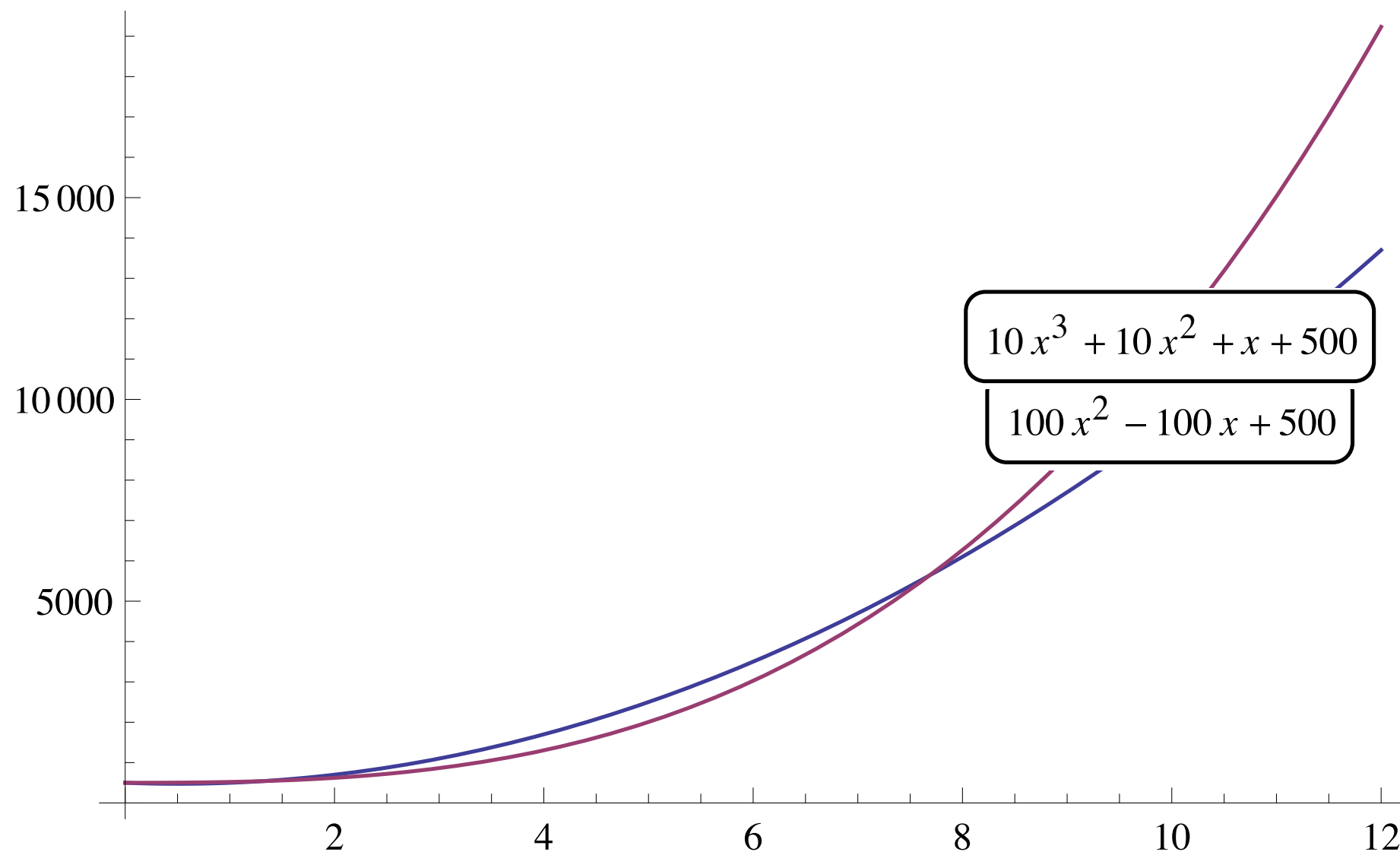


memory



Asymptotic Growth

Running time as a function of n as “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

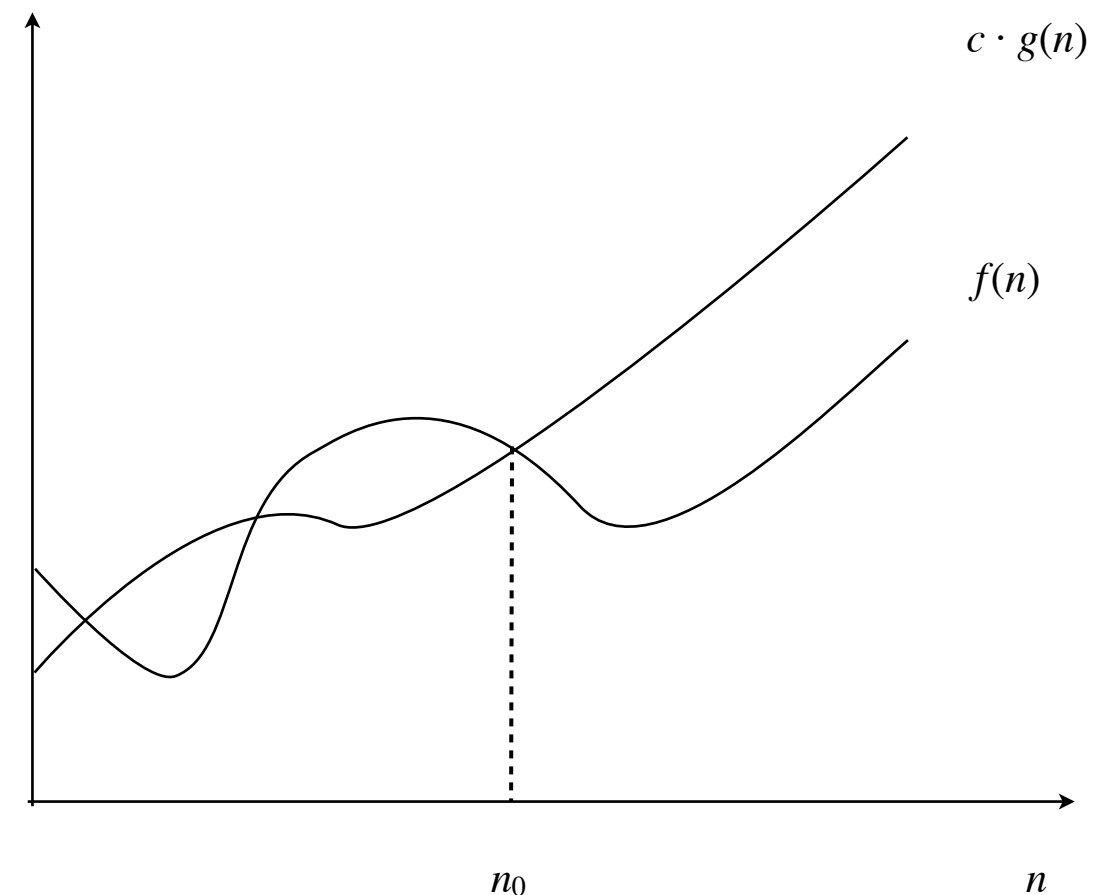
$$0 \leq f(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0$$

that is, for sufficiently large n , $f(n)$ is asymptotically **bounded above** by $g(n)$

Examples

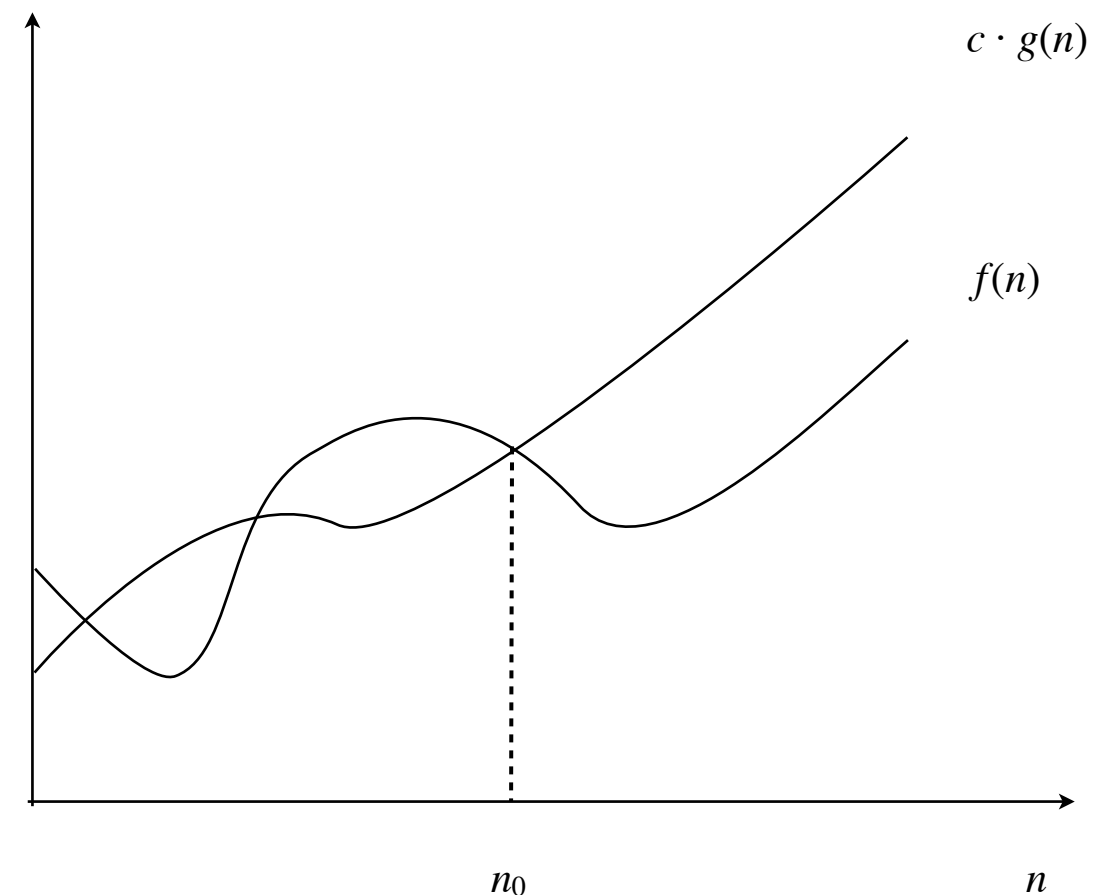
- $100n^2 = O(n^2)$
- $n \log n = O(n^2)$
- $5n^3 + 2n + 1 = O(n^3)$

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



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 they are not equal
- Okay to abuse notation in this way



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.

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- Ω 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 $n_0 \geq 0$

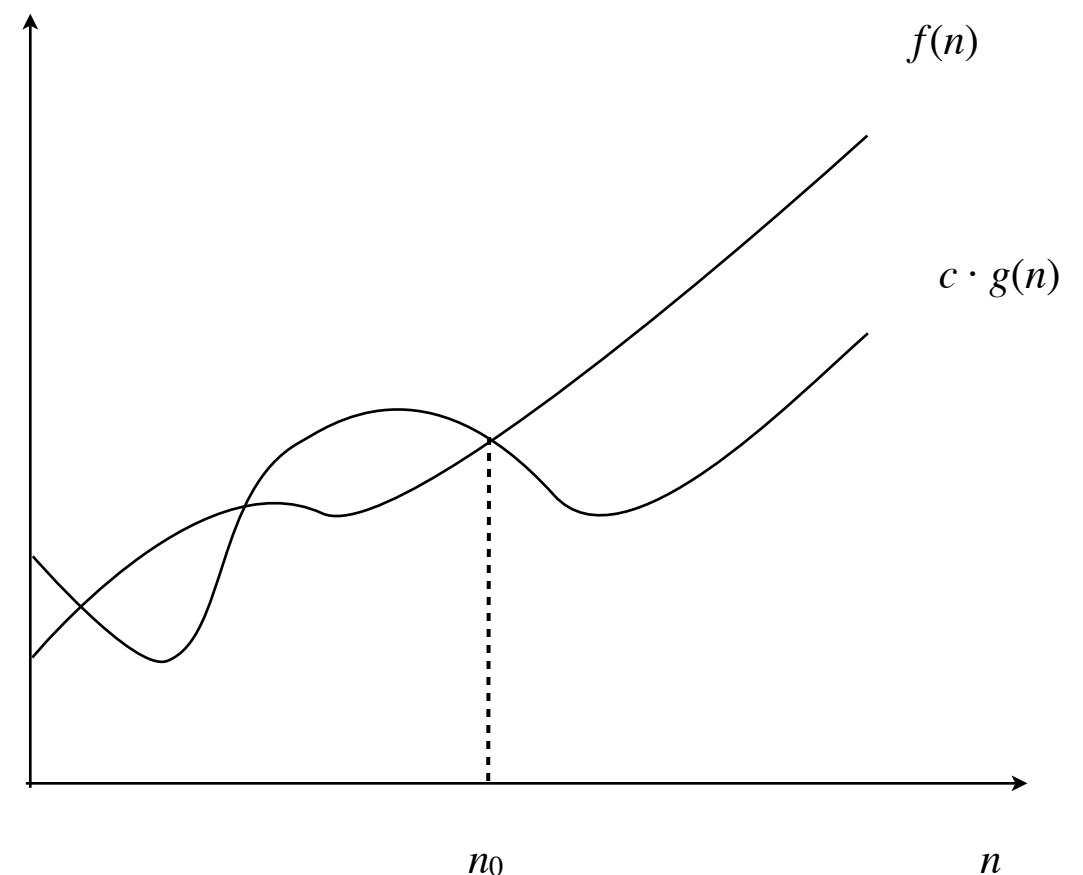
$$f(n) \geq c \cdot g(n) \geq 0 \quad \text{for all } n \geq n_0$$

That is, for sufficiently large n , $f(n)$ is asymptotically bounded below by $g(n)$

Examples

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

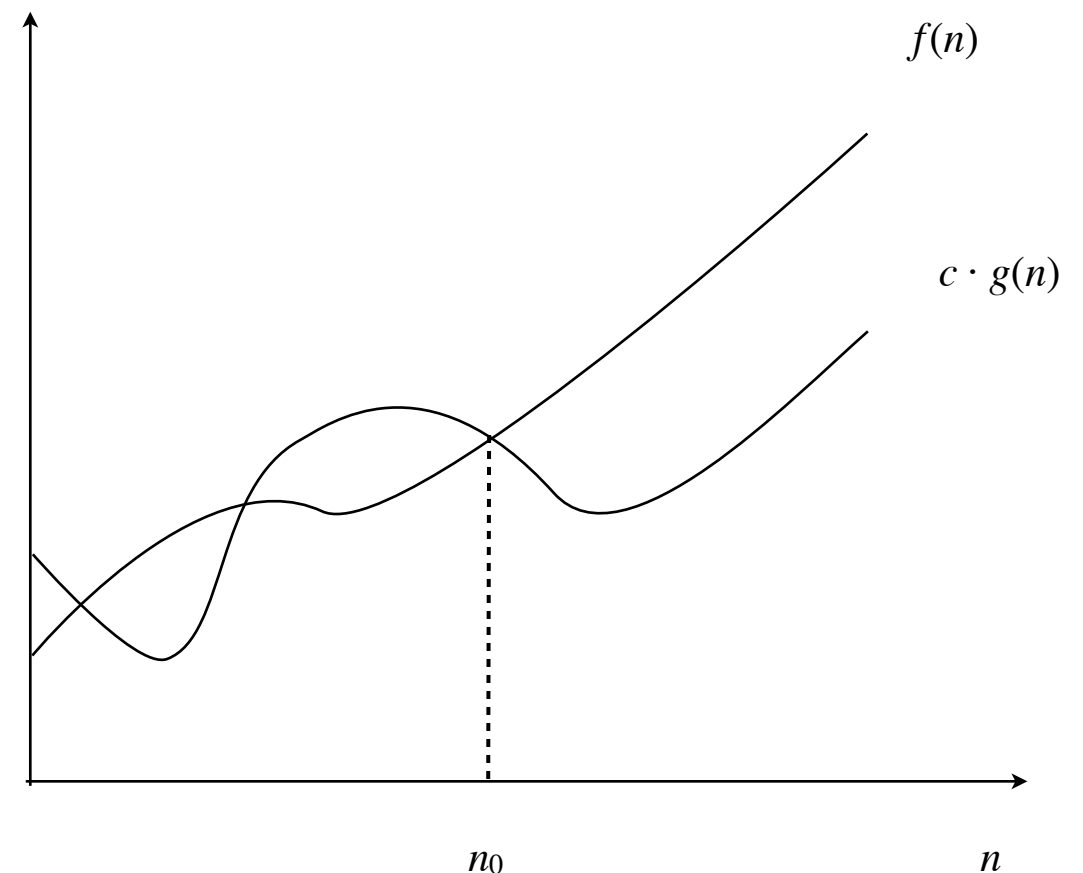
Typical usage. Insertion sort makes $\Omega(n^2)$ comparisons in the worst case



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
- Insertion sort take $\Omega(n^2)$ steps in some cases
- Mergesort takes $\Omega(n \log n)$ steps in all cases
- will cover this in a few weeks



Asymptotically Tight Bounds

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

$$f(n) = O(g(n)) \quad \text{and} \quad f(n) = \Omega(g(n))$$

Examples

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

