Objectives

- Review: Asymptotic running times
- Classes of running times
- Implementing Gale-Shapley algorithm

Review Asymptotic Bounds

- How do we define “efficient”?
- What does O(f(n)) mean?
  - How do we know if a function $\in$ O(f(n))?
- What are the other bounds we discussed?

Review: Asymptotic Order of Growth: Upper Bounds

- **T(n)** is the worst case running time of an algorithm

"order f(n)"

- We say that T(n) is **O(f(n))** if there exist constants

c cannot depend on n

sufficiently large n

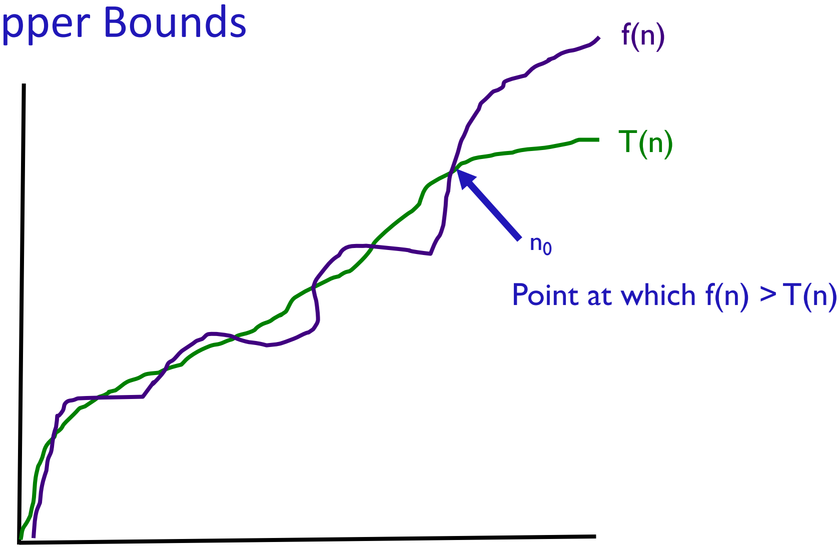
$c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have

$T(n) \leq c \cdot f(n)$

T(n) is bounded above by a constant multiple of f(n)

→T is ***asymptotically upperbounded*** by f

Review: Asymptotic Order of Growth: Upper Bounds



Asymptotic: what happens as input size grows to infinity

Review: Upper Bounds Example

- $T(n) = pn^2 + qn + r$
  - p, q, r are positive constants
- For all $n \geq 1$,

$$\begin{aligned} T(n) &= pn^2 + qn + r \\ &\leq pn^2 + qn^2 + rn^2 \\ &= (p+q+r)\, n^2 \\ &= c\, n^2 \end{aligned}$$

→ $T(n) \leq cn^2$, where $c = p+q+r$

→ $T(n) \in O(n^2)$

- Also correct to say that $T(n) \in O(n^3)$

Justification to ignore the lower-order terms

Review: Asymptotic Order of Growth: Lower Bounds

- Complementary to upper bound

ε cannot depend on n

- T(n) is **Ω(f(n))** if there exist constants $\varepsilon > 0$ and

sufficiently large n

$n_0 \geq 0$ such that for all $n \geq n_0$, we have

$T(n) \geq \varepsilon \cdot f(n)$

T(n) is bounded below by a constant multiple of f(n)

→T is ***asymptotically lowerbounded*** by f

Review: Lower Bounds Example

- $T(n) = pn^2 + qn + r$
  - p, q, r are positive constants
- Idea: *Deflate* terms rather than inflate
- For all $n \geq 0$,

$$T(n) = pn^2 + qn + r \geq pn^2$$
$$\rightarrow T(n) \geq \varepsilon n^2, \text{ where } \varepsilon = p > 0$$
$$\rightarrow T(n) \in \Omega(n^2)$$

- Also correct to say that $T(n) \in \Omega(n)$

Review: Tight bounds

T(n) is **Θ(f(n))** if T(n) is both O(f(n)) and Ω(f(n))

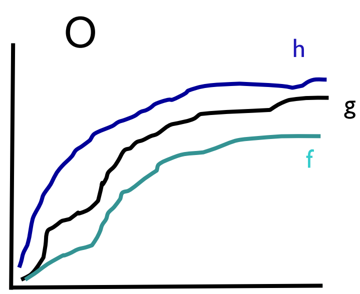
- The “right” bound

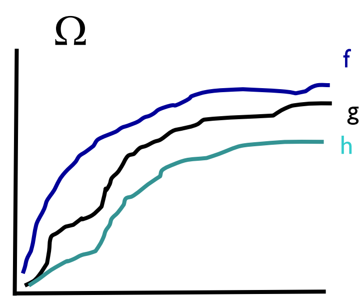
Property: Transitivity

How is this property helpful to us when analyzing algorithm runtimes?

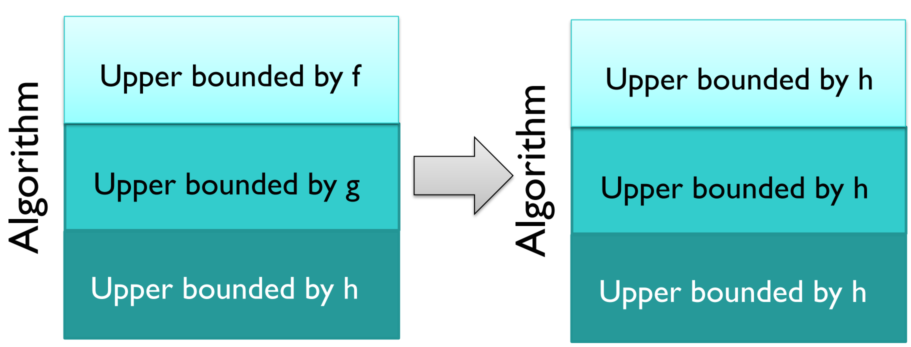
- If f = O(g) and g = O(h), then f = O(h)
- If f = Ω(g) and g = Ω(h), then f = Ω(h)
- If f = Θ(g) and g = Θ(h), then f = Θ(h)

Proofs in book





Applying Transitivity Property in Algorithm Analysis



Transitivity property: If f = O(g) and g = O(h), then f = O(h)

Property: Additivity

How is this property helpful to us when analyzing algorithm runtimes?

- If f = O(h) and g = O(h), then f + g = O(h)
- If f = Ω(h) and g = Ω(h), then f + g = Ω(h)
- If f = Θ(h) and g = Θ(h), then f + g = Θ(h)

Proofs in book

Sketch proof for O:

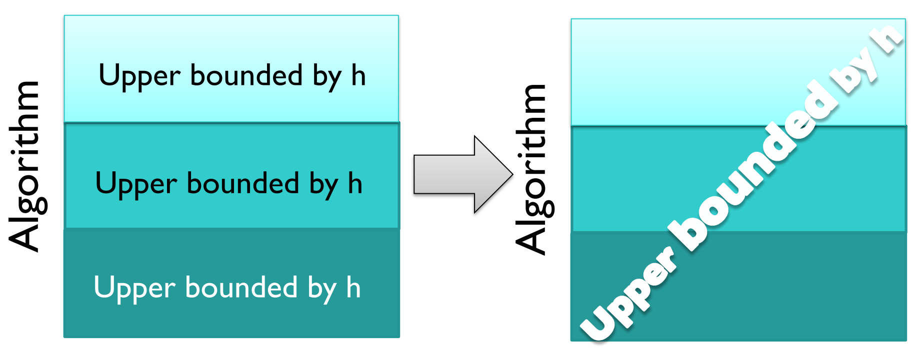
By defn, $f \leq c \cdot h$

By defn, $g \leq d \cdot h$

$f + g \leq c \cdot h + d \cdot h = (c + d)\, h = c' \cdot h$

→ f + g is O(h)

Applying Additivity Property in Algorithm Analysis



Additivity property:
If f = O(h) and g = O(h), then f + g = O(h)

Practice: Asymptotic Order of Growth

What are the upper bounds, lower bounds, and tight bound on T(n)?

- $T(n) = 32n^3 + 17n + 32$

Practice: Asymptotic Order of Growth

- $T(n) = 32n^3 + 17n + 32$
  - $T(n) \in$
    - $O(n^3)$, $O(n^4)$
    - $\Omega(n^3)$, $\Omega(n)$
    - $\Theta(n^3)$
  - T(n) is **not** $O(n)$, $\Omega(n^4)$, $\Theta(n)$, or $\Theta(n^2)$

ASYMPTOTIC BOUNDS FOR CLASSES OF ALGORITHMS

Asymptotic Bounds for Polynomials

- $a_0 + a_1 n + \ldots + a_d n^d \in \Theta(n^d)$ if $a_d > 0$

  ➔ Runtime determined by highest-order term

- Polynomial time. Running time is $O(n^d)$ for some constant *d* that is independent of the input size *n*
- Other examples of polynomial times:
  - $O(n^{1/2})$
  - $O(n^{1.58})$
  - $O(n \log n) \leq O(n^2)$

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$
  - Approximate: To represent *n* in base-*b*, need *x*+1 digits

N	b	x
100	10	
1000	10	
100	2	
1000	2	

Jan 14, 2019 Sprenkle - CSCI211 17

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$
  - Approximate: To represent *n* in base-*b*, need *x*+1 digits

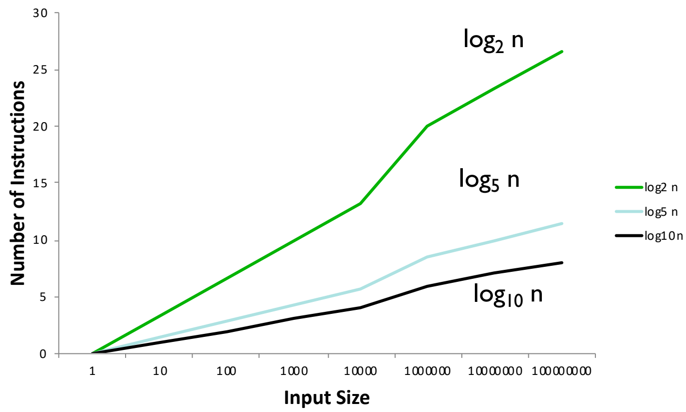
N	b	x
100	10	2
1000	10	3
100	2	6.64
1000	2	9.92

Describe the running time of an O(log n) algorithm as the input size grows. Compare with polynomials.

Jan 14, 2019 Sprenkle - CSCI211 18

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$
  - x is number of digits to represent n in base-b representation



Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$

  → Slowly growing functions

- Identity: $\log_a n = \log_b n / \log_b a$
  - Means that

    $\log_a n = \underbrace{1/\log_b a}_{\text{Constant!}} * \log_b n$

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

Asymptotic Bounds for Logarithms

- Logarithms. $\log_b n = x$, where $b^x = n$

➔ Slowly growing functions

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

➔ Don't need to specify the base

- For every x > 0, $\log n = O(n^x)$

➔ Log grows slower than every polynomial

Asymptotic Bounds for Exponentials

- Exponentials: functions of the form $f(n) = r^n$ for constant base *r*
  - Faster growth rates as *n* increases

- For every r > 1 and every d > 0, $n^d = O(r^n)$

➔ Every exponential grows faster than every polynomial

Summary of Asymptotic Bounds

- In terms of growth rates ....

Logarithms < Polynomials < Exponentials

- Practice comparing functions on next problem set
  - See Chapter 2 solved exercise

Review: Our Process

1. Understand/identify problem
   - Simplify as appropriate
2. Design a solution
3. Analyze
   - Correctness, efficiency
   - May need to go back to step 2 and try again
4. Implement
   - Within bounds shown in analysis

IMPLEMENTING GALE-SHAPLEY ALGORITHM

Review: Gale-Shapley Stable Matching Algorithm

```
Initialize each person to be free
while (some man is free and hasn't proposed to every woman)
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged and m' to be free
    else
        w rejects m
```

How Can We Implement The Algorithm Efficiently?

- What is our goal for the implementation's runtime?
- What do we need to model?
- How should we represent them?

How Can We Implement The Algorithm Efficiently?

- What is our goal for the implementation's runtime?
  - $O(N^2)$
- What do we need to model?
- How should we represent them?

Stable Matching Implementation

- What do we need to represent?
- How should we represent them?

Data	How represented
Men, Women	
Preference lists	
Unmatched men	
Who men proposed to	
Engagements	

What's the difference between an array and a list?

Arrays

- *Fixed* number of elements
- What is the runtime of
  - Determining the value of the i^{th} item in the array?
  - Determining if a value *e* is in the array?
  - Determining if a value *e* is in the array if the array is sorted?

Array Operations' Running Times

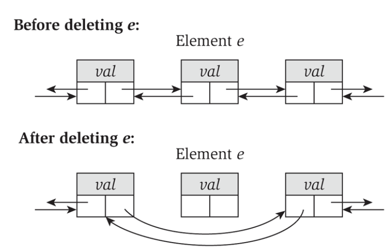
Operation	Running Time
Value of i^{th} item	O(1) → direct access
If *e* is in the array	O(n) → look through all the elements
If *e* is in the array if sorted	O(log n) → binary search

Limitation of arrays?

Fixed size, so difficult to add/delete elements

Lists

- Dynamic set of elements
  - Linked list
  - Doubly linked list
- What is the running time to
  - Add an element to the list?
  - Delete an element from the list?
  - Find an element *e* in the list?
  - Find the i^{th} element in the list?



List Operations' Running Time

Operation	Running Time
Add element	O(1)
Delete element	O(1)
Find element	O(n)
Find i^{th} element	O(i)

Disadvantage of list instead of array?

Finding i^{th} element is slower

Converting between Lists and Arrays (and Vice Versa)

- What is the running time of converting a list to an array?
- An array to a list?

O(n)

Looking Ahead

- Wiki due tonight at midnight
  - 1st two pages of preface
  - 1.1
  - 2.1, 2.2
- Problem Set 1 due Friday, before class