UNIVERSITY OF NORTH TEXAS

CSCE 3110
Data Structures and Algorithms

Algorithm Analysis I (cont.)

Reading: Weiss, chap. 2

Credits of some slides go to S. Skiena at SUNY Stony Brook.

Asymptotic Notation

- Asymptotic notation
  - Big Oh
  - Big Omega
  - Big Theta
  - Little Oh
  - Little Omega

Asymptotic Notation - little oh, o

$$o\big(g(n)\big) = \{f(n): \boxed{\forall} c > 0, \exists n_0\ s.t. \forall n \geq n_0 : 0 \leq f(n) \boxed{<} cg(n)\}$$

- In plain English: $o(g(n))$ are all functions $f(n)$ for which for all c>0, there exists a constant n_0>0 such that for all $n \geq n_0$, $0 \leq f(n) < cg(n)$
  - $g(n)$ is an asymptotic **<u>upper bound</u>** for $f(n)$, but not tight
  - $f(n)$ becomes insignificantly relative to $g(n)$ as n grows
  - $f(n)$ grows asymptotically slower than $g(n)$

Asymptotic Notation - little oh, o

$$o(g(n)) = \{f(n): \boxed{\forall} c > 0, \exists n_0 \; s.t. \forall n \geq n_0 : 0 \leq f(n) \boxed{<} cg(n)\}$$

- In plain English: $o(g(n))$ are all functions $f(n)$ for which for all c>0, there exists a constant n_0>0 such that for all $n \geq n_0$, $0 \leq f(n) < cg(n)$
  - $g(n)$ is an asymptotic **upper bound** for $f(n)$, but not tight
  - $f(n)$ becomes insignificantly relative to $g(n)$ as n grows
  - $f(n)$ grows asymptotically slower than $g(n)$

- Similar to $O(g(n))$, intuitively, you can think of o as "<" for functions

Asymptotic Notation - little oh, o

- Examples
  - $n^{1.999} = o(n^2)$?
  - $n^2/\log(n) = o(n^2)$?
  - $n^2 = o(n^2)$?
  - $n^2/1000 = o(n^2)$?

Asymptotic Notation - little oh, o

- Examples
  - $n^{1.999} = o(n^2)$
  - $n^2/\log(\text{n}) = o(n^2)$
  - $n^2 \neq o(n^2)$
  - $n^2/1000 \neq o(n^2)$

Asymptotic Notation - little omega, ω

$$\omega(g(n)) = \{f(n): \forall c > 0, \exists n_0\ s.t. \forall n \geq n_0: 0 \leq cg(n) < f(n)\}$$

- In plain English: $\omega(g(n))$ are all functions $f(n)$ for which for all c>0, there exists a constant n_0>0 such that for all $n \geq n_0$, $0 \leq cg(n) < f(n)$
  - — $\omega(n)$ is an asymptotic **lower bound** for $f(n)$, but not tight
  - — $f(n)$ becomes arbitrarily large to $g(n)$ as n grows
  - — $f(n)$ grows asymptotically faster than $g(n)$

Asymptotic Notation - little omega, ω

$$\omega\big(g(n)\big) = \{f(n)\colon \forall c > 0, \exists n_0 \; s.t. \forall n \geq n_0\colon 0 \leq cg(n) < f(n)\}$$

- In plain English: $\omega(g(n))$ are all functions $f(n)$ for which for all c>0, there exists a constant n_0>0 such that for all $n \geq n_0$, $0 \leq cg(n) < f(n)$
  - $\omega(n)$ is an asymptotic **lower bound** for $f(n)$, but not tight
  - $f(n)$ becomes arbitrarily large to $g(n)$ as n grows
  - $f(n)$ grows asymptotically faster than $g(n)$

- Similar to $\Omega(g(n))$, intuitively, you can think of ω as ">" for functions

Asymptotic Notation - little omega, ω

- Examples
  - $n^{2.0001} = \omega(n^2)$?
  - $n^2 \log(n) = \omega(n^2)$?
  - $n^2 = \omega(n^2)$?

Asymptotic Notation - little omega, ω

- Examples
  - $n^{2.0001} = \omega(n^2)$
  - $n^2 \log(n) = \omega(n^2)$
  - $n^2 \neq \omega(n^2)$

Asymptotic Notation

- When using asymptotic notations
  - Drop lower-order terms
  - Ignore constant coefficient in the leading term

Asymptotic Notation

- When using asymptotic notations
  - Drop lower-order terms
  - Ignore constant coefficient in the leading term

- Asymptotic notation is a way to compare functions
  - $O \approx \leq$
  - $\Omega \approx \geq$
  - $\Theta \approx =$
  - $o \approx <$
  - $\omega \approx >$

Asymptotic Notation Addition

Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.

Q1: What do we know about $g'(n) = f(n) + g(n)$?

Q2: What do we know about the lower bounds on g'?

Asymptotic Notation Addition

Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.

Q1: What do we know about $g'(n) = f(n) + g(n)$?
A1: Adding the bounding constants shows $g'(n) = O(n^2)$.

Q2: What do we know about the lower bounds on g'?
A2: We know nothing about the lower bounds on g' because we know nothing about lower bounds on f and g.

Asymptotic Notation Multiplication by Constant

- Multiplication by a constant does not change the asymptotic
  - $O(c \cdot f(n)) \to O(f(n))$
  - $\Omega(c \cdot f(n)) \to \Omega(f(n))$
  - $\Theta(c \cdot f(n)) \to \Theta(f(n))$

The "old constant" C from the Big Oh becomes $c \cdot C$.

Asymptotic Notation Multiplication by Function

- When both functions in a product are increasing, both are important
  - $O(f(n)) \cdot O(g(n)) \rightarrow O(f(n) \cdot g(n))$
  - $\Omega(f(n)) \cdot \Omega(g(n)) \rightarrow \Omega(f(n) \cdot g(n))$
  - $\Theta(f(n)) \cdot \Theta(g(n)) \rightarrow \Theta(f(n) \cdot g(n))$

This is why the running time of two nested loops is $O(n^2)$.

Selection Sort

```
void selection_sort(item_type s[], int n) {
    int i, j;      /* counters */
    int min;       /* index of minimum */

    for (i = 0; i < n; i++) {
        min = i;
        for (j = i + 1; j < n; j++) {
            if (s[j] < s[min]) {
                min = j;
            }
        }
        swap(&s[i], &s[min]);
    }
}
```

Worst Case Analysis

- The outer loop goes around n times.
- The inner loop goes around at most n times for each iteration of the outer loop
- Thus, selection sort takes at most $n \times n \rightarrow O(n^2)$ time in the worst case.

Asymptotic Dominance in Action

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Implications of Dominance

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

- Exponential algorithms get hopeless fast.
- Quadratic algorithms get hopeless at or before 1,000,000.
- $O(n \log n)$ is possible to about one billion.
- $O(\log n)$ never sweats

Testing Dominance

$f(n)$ **dominates** $g(n)$ **if** $\lim_{n \to \infty} g(n)/f(n) = 0$**, which is the same as saying** $g(n) = o(f(n))$.

Note the little-oh – it means "grows strictly slower than"

Properties of Dominance

- n^a dominates n^b if $a > b$ since

$$\lim_{n \to \infty} n^b / n^a = n^{b-a} \to 0$$

Properties of Dominance

- n^a dominates n^b if $a > b$ since

$$\lim_{n\to\infty} n^b/n^a = n^{b-a} \to 0$$

- $n^a + o(n^a)$ doesn't dominate n^a since

$$\lim_{n\to\infty} n^a/(n^a + o(n^a)) \to 1$$

Dominance Rankings

- You must come to accept the dominance ranking of the basic functions:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

Advanced Dominance Rankings

- Additional functions arise in more sophisticated analysis than we will do in this course :

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg \log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

Logarithms

- It is important to understand deep in your bones what logarithms are and where they come from.
- A logarithm is simply an inverse exponential function.
- Saying $b^x = y$ is equivalent to saying that $x = \log_b y$.
- Logarithms reflect how many times we can double something until we get to n or halve something until we get to 1

Logarithms

In binary search we throw away half the possible number of keys after each comparison. Thus twenty comparisons suffice to find any name in the million-name Manhattan phone book!

How many time can we halve n before getting to 1?

Answer: $\lceil \lg n \rceil$.

Logarithms

In binary search we throw away half the possible number of keys after each comparison. Thus twenty comparisons suffice to find any name in the million-name Manhattan phone book!

How many time can we halve n before getting to 1?

Logarithms and Trees

How tall a binary tree do we need until we have n leaves?
The number of potential leaves doubles with each level.
How many times can we double 1 until we get to n?

Logarithms and Trees

How tall a binary tree do we need until we have n leaves?
The number of potential leaves doubles with each level.
How many times can we double 1 until we get to n?
Answer: $\lceil \lg n \rceil$.

Next Class

Recurrence Relations