CSE 326: Data Structures
Lecture #6
Asymptotic Analysis

Ashish Sabharwal

Summer Quarter 2001

First a Reminder!

- Don't forget to turn-in your graded quizes with your homework!!!
- *Policy: If you don't, you are responsible for doing ALL problems as hw*

Analysis of Algorithms

- Efficiency measure
  - – how long the program runs time complexity
  - – how much memory it uses space complexity
    - For today, we'll focus on time complexity only
- *Why analyze at all?*
  - – Confidence: algorithm will work well in practice
  - – Insight : alternative, better algorithms

Time Complexity

- We count number of abstract steps
  - – Not physical runtime in seconds
  - – Not every machine instruction
- What is one abstract step?

  count = count + 1

  $y = a*x^3 + b*x + c$

  if (n ≥ 2 sqrt(m) || n ≤ 0.5 sqrt(m)) …

Asymptotic Analysis

- Complexity as a function of input size n
  - $T(n) = 4n + 5$
  - $T(n) = 0.5\ n \log n - 2n + 7$
  - $T(n) = 2^n + n^3 + 3n$
- *What happens as n grows?*

Why do we care?

- Most algorithms are fast for small n
  - Time difference too small to be noticeable
  - External things dominate (OS, disk I/O, …)
- n is typically large in practice
  - Databases, internet, graphics, …
- Time difference really shows up as n grows!

Rates of Growth

- Suppose we can execute 10^{10} ops / sec

n=? / T(n)=?	10	100	1,000	10,000
n	10^{-9}s	10^{-8}s	10^{-7}s	10^{-6}s
$n \log_2 n$	10^{-9}s	10^{-8}s	10^{-6}s	10^{-5}s
n^2	10^{-8}s	10^{-6}s	10^{-4}s	10^{-2}s
n^3	10^{-7}s	10^{-4}s	0.1s	100s
2^n	10^{-7}s	10^{20}s	10^{291}s	*forever!*

10^4s = 2.8 hrs 10^{18}s = 30 billion years

Obtaining Asymptotic Bounds

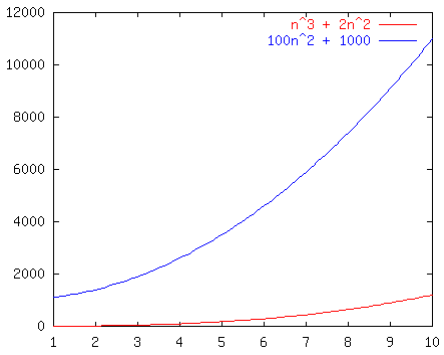
- Eliminate low order terms
  - $4n + 5 \Rightarrow 4n$
  - $0.5\ n \log n - 2n + 7 \Rightarrow 0.5\ n \log n$
  - $2^n + n^3 + 3n \Rightarrow 2^n$
- Eliminate coefficients
  - $4n \Rightarrow n$
  - $0.5\ n \log n \Rightarrow n \log n$
  - $n \log n^2 = 2\ n \log n \Rightarrow n \log n$

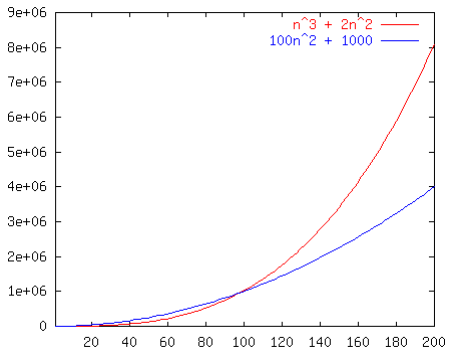
Race Against Time!

Race #	$T_1(n)$	$T_2(n)$	Which is faster?
1	$n^3 + 2n^2$	$100n^2 + 1000$	
2	$n^{0.1}$	$\log n$	
3	$n + 100n^{0.1}$	$2n + 10 \log n$	
4	$5n^5$	$n!$	
5	$n^{-15}2^n/100$	$1000n^{15}$	
6	$8^{2\log n}$	$3n^7 + 7n$	
7	mn^3	$2^m n$	

Race 1

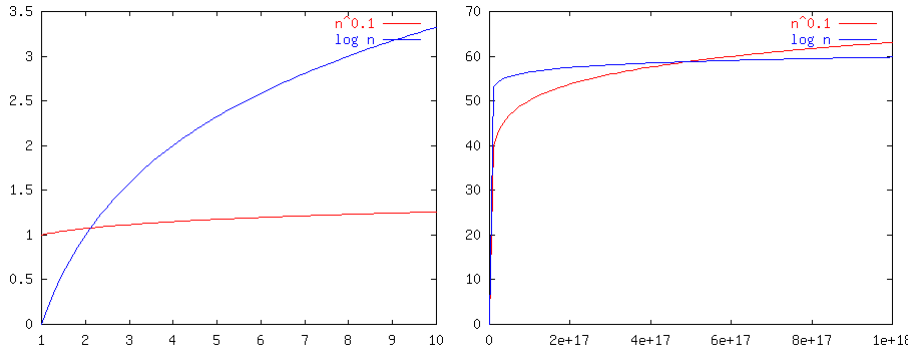
$n^3 + 2n^2$ vs. **$100n^2 + 1000$**





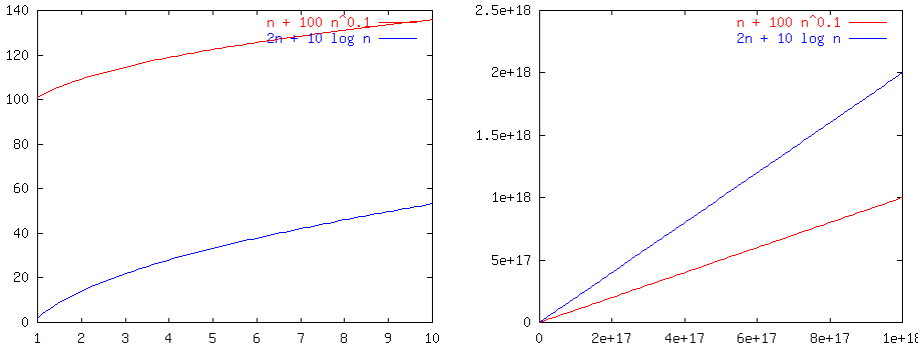
Race 2

$n^{0.1}$ vs. **log n**



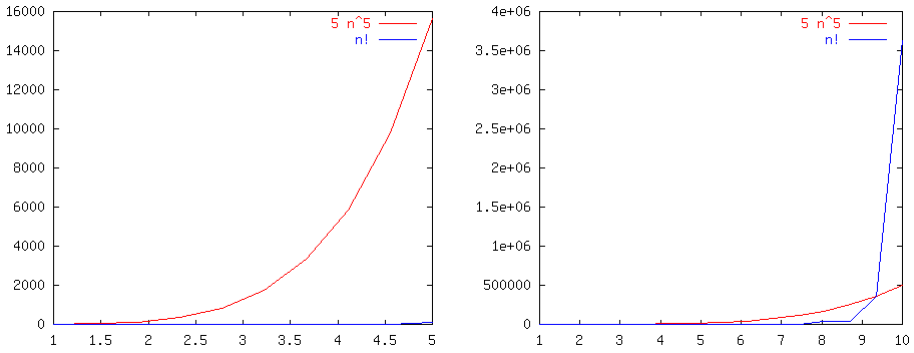
Race 3

$n + 100n^{0.1}$ vs. **2n + 10 log n**



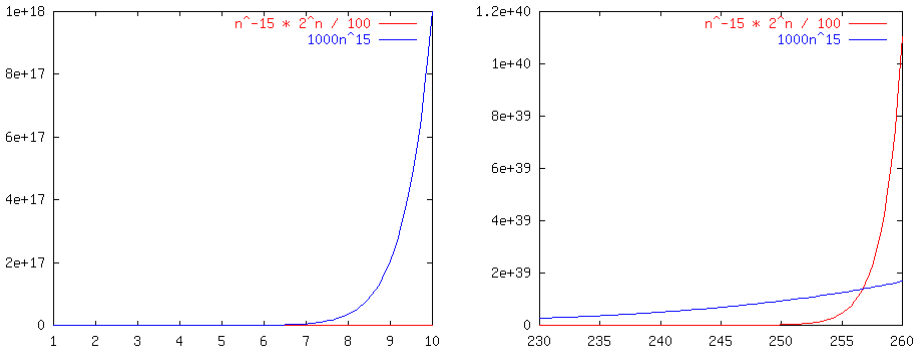
Race 4

$5n^5$ vs. **$n!$**



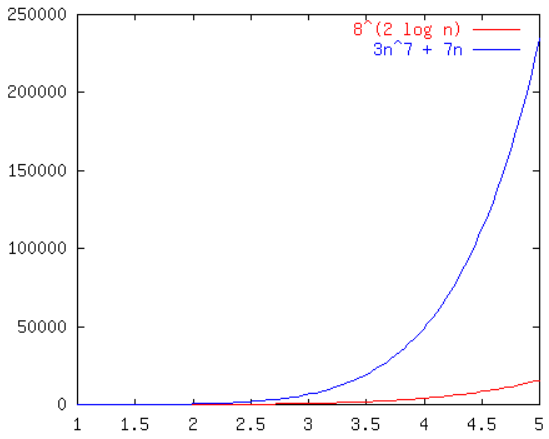
Race 5

$n^{-15}2^n/100$ vs. **$1000n^{15}$**



Race 6

$8^{2\log(n)}$ vs. $3n^7 + 7n$



Race Against Time! (2)

Race #	$T_1(n)$	$T_2(n)$	Which is faster?
1	$n^3 + 2n^2$	$100n^2 + 1000$	T_2 : $O(n^2)$
2	$n^{0.1}$	$\log n$	T_2 : $O(\log n)$
3	$n + 100n^{0.1}$	$2n + 10 \log n$	Tie: $O(n)$
4	$5n^5$	$n!$	T_1 : $O(n^5)$
5	$n^{-15}2^n/100$	$1000n^{15}$	T_2 : $O(n^{15})$
6	$8^{2\log n}$	$3n^7 + 7n$	T_1 : $O(n^6)$
7	mn^3	$2^m n$	*It depends!*

Typical Growth Rates

– constant:	$O(1)$	
– logarithmic:	$O(\log n)$	($\log_k n$, $\log n^2 \in O(\log n)$)
– poly-log:	$O(\log^k n)$	
– linear:	$O(n)$	
– log-linear:	$O(n \log n)$	
– superlinear:	$O(n^{1+c})$	(c is a constant > 0)
– quadratic:	$O(n^2)$	
– cubic:	$O(n^3)$	
– polynomial:	$O(n^k)$	(k is a constant)
– exponential:	$O(c^n)$	(c is a constant > 1)

Terminology

- $T(n) \in O(f(n))$
  - $\exists$ constants c and n_0 s.t. $T(n) \leq c\, f(n) \quad \forall n \geq n_0$
  - $1, \log n, n, 100n \in O(n)$
- $T(n) \in \Omega(f(n))$
  - $\exists$ constants c and n_0 s.t. $T(n) \geq c\, f(n) \quad \forall n \geq n_0$
  - $n/10, n^2, 100 \cdot 2^n, n^3 \log n \in \Omega(n)$
- $T(n) \in \theta(f(n))$
  - $T(n) \in O(f(n))$ and $T(n) \in \Omega(f(n))$
  - $n+4, 2n, 100n, 0.01\, n + \log n \in \theta(n)$

Terminology (2)

- $T(n) \in o(f(n))$
  - $T(n) \in O(f(n))$ and $T(n) \notin \theta(f(n))$
  - $1, \log n, n^{0.99} \in o(n)$
- $T(n) \in \omega(f(n))$
  - $T(n) \in \Omega(f(n))$ and $T(n) \notin \theta(f(n))$
  - $n^{1.01}, n^2, 100 \cdot 2^n, n^3 \log n \in \omega(n)$

Terminology (3)

- Roughly speaking, the correspondence is

O	$\leq$
Ω	$\geq$
θ	$=$
o	$<$
ω	$>$

Types of Analysis

Three orthogonal axes:

- bound flavor
  - upper bound (O, o)
  - lower bound (Ω, ω)
  - asymptotically tight (θ)
- analysis case
  - worst case (adversary)
  - average case
  - best case
  - "common" case
- analysis quality
  - loose bound (most true analyses)
  - tight bound (no better bound which is asymptotically different)

Analyzing Code

- General guidelines

Simple C++ operations	- constant time
consecutive stmts	- sum of times per stmt
conditionals	- sum of branches and condition
loops	- sum over iterations
function calls	- cost of function body

Simple loops

```
sum = 0
for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1
```

Simple loops (2)

```
sum = 0
for i = 1 to n do
  for j = i to n do
    sum = sum + 1
```

Conditionals and While Loop

- Conditional

  if c **then** s_1 **else** s_2

- Loops

  while c **do** s

Recursion

- Recursion
  - Almost always yields a *recurrence*
  - Recursive max
- Example: Factorial

```
fac(n)
  if n = 0 return 1
  else return n * fac(n - 1)
```

$T(0) = 1$

$T(n) \leq c + T(n - 1)$ *if* $n > 0$

Example: Factorial

Analysis by simple calculation

$T(n) \leq c + c + T(n - 2)$
(by substitution)

$T(n) \leq c + c + c + T(n - 3)$
(by substitution, again)

$T(n) \leq kc + T(n - k)$
(extrapolating $0 < k \leq n$)

$T(n) \leq nc + T(0) = nc + b$
(setting $k = n$)

- $T(n) \in$

Example: Mergesort

- Mergesort algorithm
  - If list has 1 element, return
  - Otherwise split list in half, sort first half, sort second half, merge together

- $T(1) = 1$

  $T(n) \leq 2T(n/2) + cn$ *if* $n > 1$

Sorting the two halves recursively

Splitting and merging

Example: Mergesort (2)

Analysis by simple calculation

$$
\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&\le 2(2T(n/4) + c(n/2)) + cn \\
&= 4T(n/4) + cn + cn \\
&\le 4(2T(n/8) + c(n/4)) + cn + cn \\
&= 8T(n/8) + cn + cn + cn \\
&\le 2^k T(n/2^k) + kcn \quad \text{(extrapolating } 1 < k \le n\text{)} \\
&\le nT(1) + cn \log n \quad \text{(for } 2^k = n \text{ or } k = \log n\text{)}
\end{aligned}
$$

- $T(n) \in ?$

Example: Mergesort (3)

Analysis by induction

1. Guess the answer!
   $T(n) \le an \log n + b$
   a, b constants, not known yet
2. Verify base case
   $T(1) \le b$ $\quad b \ge 1$
3. Verify inductive step

$$
\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&= 2(an/2 \log n/2 + b) + cn \\
&= an \log n/2 + 2b + cn \\
&= (an \log n + b) - (an - b - cn) \\
&\le an \log n + b \quad \text{for } a > c
\end{aligned}
$$

Summary

- Determine what characterizes a problem's input size
- Express how much resources (time, memory, etc.) an algorithm requires as a function of input size using $O(\bullet)$, $\Omega(\bullet)$, $\theta(\bullet)$
  - worst case
  - best case
  - average case
  - common case