CSc 225
Algorithms and Data Structures I
Asymptotic Notations

Jianping Pan

Fall 2007

Feedback on A0

- Office hours (now official)
  - **M 3:30-4:30pm, R 1:30-2:30pm**
- "Is textbook required?" Yes
- "How much programming do we do?"
  - programming helps us to understand algorithms
- Midterm/final coverage
  - lectures, tutorials, assignments, required textbook reading
- Assignments due time
  - by the end of university business hours, i.e., 4:30pm
- "Cannot hear/understand/cope"
  - if I am not clear or go too fast in class, alert me!
- Bioinformatics, AI, ... (and many algorithmic topics)
  - advanced algorithms courses in 3/4-year requiring csc225

Review

```
currentMax ← A[0]
for k ← 1 to n-1 do
   if currentMax < A[k] then
      currentMax ← A[k]
   end
end
return currentMax
```

- How has the input to be arranged to produce the best case and the worst case?

Worst case

- 1 A + 1 I +
- $\boxed{\text{1 A + (N-1)*\{ 1 A + 1 S + 1 C +}}$
  - 1 A + 1 S +
  - 1 C +
  - 1 C + 1 I +
  - 1 A + 1 I +
- }
- $\boxed{\text{1 C (to terminate loop)}}$ +
- 1 A

T(n) = 5+(n-1)*7

T(n) = 7n - 2

Q: average T(n)?

Today's topics

- Asymptotic notations
  - why asymptotic analysis
    - “In mathematics and applications, particularly the analysis of algorithms, real analysis, and engineering, asymptotic analysis is a method of describing ***limiting*** behaviour.”
    - why are we concerned about big input size?
  - Big-Oh
    - definition
    - theorem
  - Big-Omega, Big-Theta
    - little-oh, little-omega
- Special guests 3:10pm today!

Asymptotic Notation

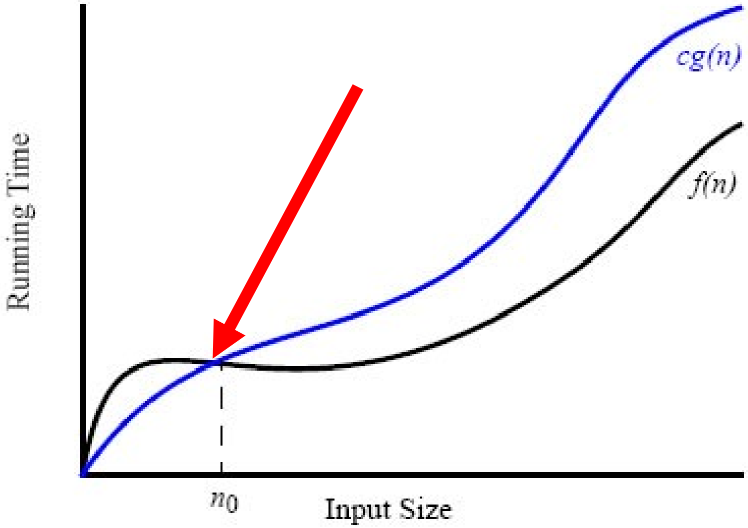
- Evaluating running time in detail as for `arrayMax` and `recursiveMax` is cumbersome
- Fortunately, there are asymptotic notations which allow us to characterize the main factors affecting an algorithm's running time without going into detail
- A good notation for ***large*** inputs

- **Big-Oh O(.)**
- Big-Omega Ω(.)
- Big-Theta Θ(.)
- Little-Oh o(.)
- Little-Omega ω(.)

slides adopted from Dr Muller's

Formal Definition of Big-Oh Notation

Let f: IN→R and g: IN→R. f(n) is O(g(n)) if and only if
there exists a real constant $c > 0$
and an integer constant $n_0 > 0$
such that $f(n) \le c \cdot g(n)$ for all $n \ge n_0$.
IN: non-negative integers
IR: real numbers



- We say
  - f(n) is *order* g(n)
  - f(n) is *big-Oh* of g(n)
  - $f(n) \in O(g(n))$
- Visually, this says that the f(n) curve must eventually fit under the cg(n) curve.

Big-Oh: Examples

- $f(n) = 4n + 20n^4 + 117$
  $O(f(n))$ is ?
- $f(n) = 1083$
  $O(f(n))$ is ?
- $f(n) = 3\log n$
  $O(f(n))$ is ?
- $f(n) = 3\log n + \log\log n$
  $O(f(n))$ is ?
- $f(n) = 2^{17}$
  $O(f(n))$ is ?
- $f(n) = 33/n$
  $O(f(n))$ is ?
- $f(n) = 2^{\log_2 n}$
  $O(f(n))$ is ?
- $f(n) = 1^n$
  $O(f(n))$ is ?

Big-Oh: Examples

- $f(n) = 4n + 20n^4 + 117$
  $O(f(n))$ is **$O(n^4)$**
  P: $4n + 20n^4 + 117 \le 90n^4$
- $f(n) = 1083$
  $O(f(n))$ is **$O(1)$**
- $f(n) = 3 \log n$
  $O(f(n))$ is **$O(\log n)$**
  P: $3 \log n \le 4 \log n$
- $f(n) = 3\log n + \log \log n$
  $O(f(n))$ is **$O(\log n)$**
  P: $3\log n + \log \log n \le 4 \log n$
- $f(n) = 2^{17}$
  $O(f(n))$ is **$O(1)$**
  P: $2^{17} \le 1\ 2^{17}$
- $f(n) = 33/n$
  $O(f(n))$ is **$O(1/n)$**
  P: $33/n \le 33(1/n)$ for $n \ge 1$
- $f(n) = 2^{\log_2 n}$
  $O(f(n))$ is **$O(n)$**
  P: $2^{\log_2 n} = 2$ by log def
- $f(n) = 1^n$
  $O(f(n))$ is **$O(1)$**
  P: $1^n = 1$ by exponential def

prove by definition

Theorem

- **R1:** If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, $a > 0$
- **R2:** If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)+e(n)$ is $O(f(n)+g(n))$
- **R3:** If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$
- **R4:** If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$
- **R5:** If $f(n) = a_0 + a_1 n + \ldots + a_d n^d$, d and a_k are constants, then $f(n)$ $O(n^d)$
- **R6:** n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$
- **R7:** $\log n^x$ is $O(\log n)$ for any fixed $x > 0$
- **R8:** $\log^x n$ is $O(n^y)$ for any fixed constants $x > 0$ and $y > 0$

prove by using theorem

Names of Most Common Big Oh Functions

- Constant $O(1)$
- Logarithmic $O(\log n)$
- Linear $O(n)$
- Quadratic $O(n^2)$
- Polynomial $O(n^k)$ k is a constant

- Exponential $O(2^n)$
- Exponential $O(a^n)$ a is a constant and $a > 1$

Functions Ordered by Growth and Rate

n	log n	n	n log n	n^2	n^3	2^n	n!
10	3.3	10	33	10^2	10^3	10^3	10^6
10^2	6.6	10^2	6.6 10^2	10^4	10^6	10^{30}	10^{158}
10^3	10	10^3	1.0 10^3	10^6	10^9		
10^4	13	10^4	1.3 10^4	10^8	10^{12}		
10^5	17	10^5	1.7 10^5	10^{10}	10^{15}		
10^6	20	10^6	2.0 10^6	10^{12}	10^{18}		

Assume a computer executing 10^{12} operations per second.
To executive 2^{100} operations takes 4 10^{10} years.
To executive 100! operations takes much longer still.

Q: how about 2*10^12 ops?

L'Hôpital's Rule

$$\lim_{n \to b} \frac{f_1(n)}{f_2(n)} = \lim_{n \to b} \frac{f_1'(n)}{f_2'(n)} = \begin{cases} 0, \text{ then } f_2(n) \text{ grows faster} \\ 0 < x < \infty, \text{ then inconclusive} \\ \infty, \text{ then } f_1(n) \text{ grows faster} \end{cases}$$

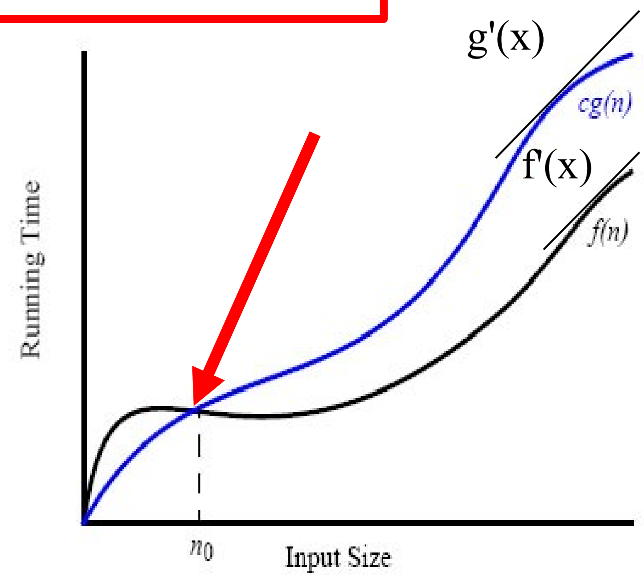
Example

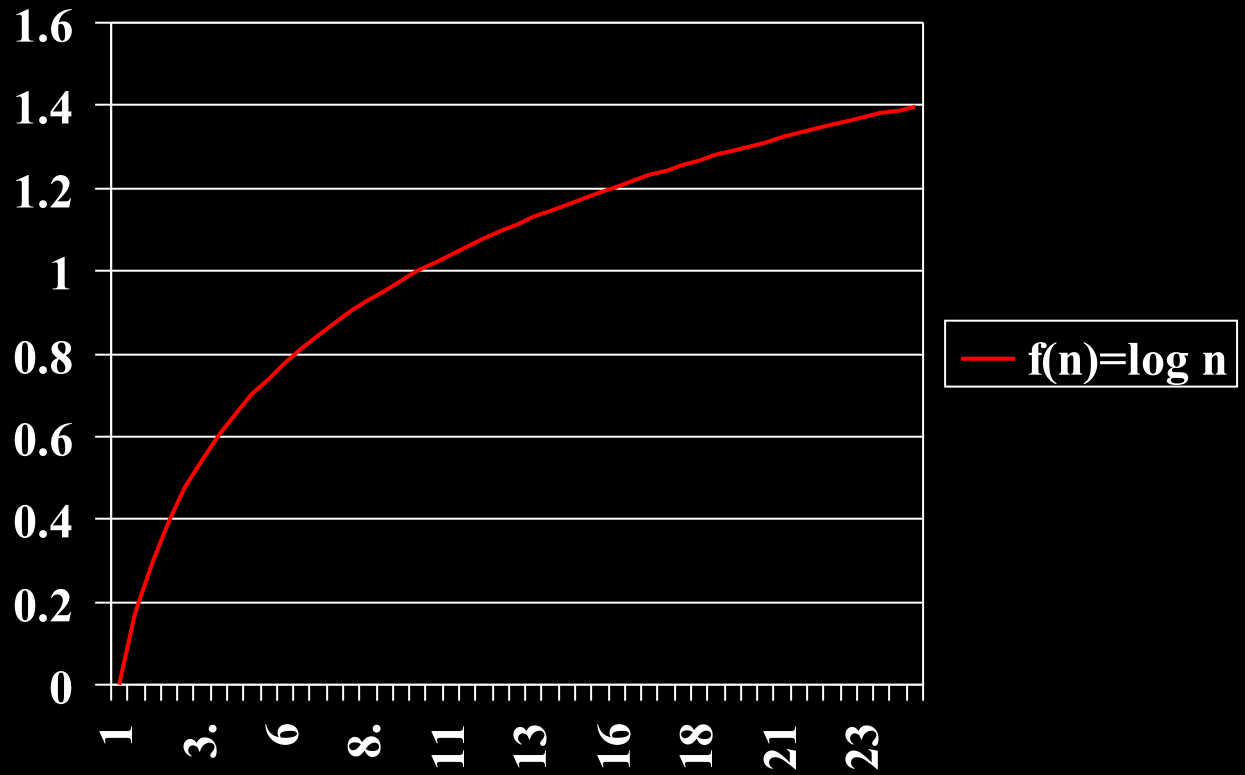
$f_1(n) = n^2 \quad f_1'(n) = 2\text{n}$

$f_2(n) = e^n \quad f_2'(n) = e^n$

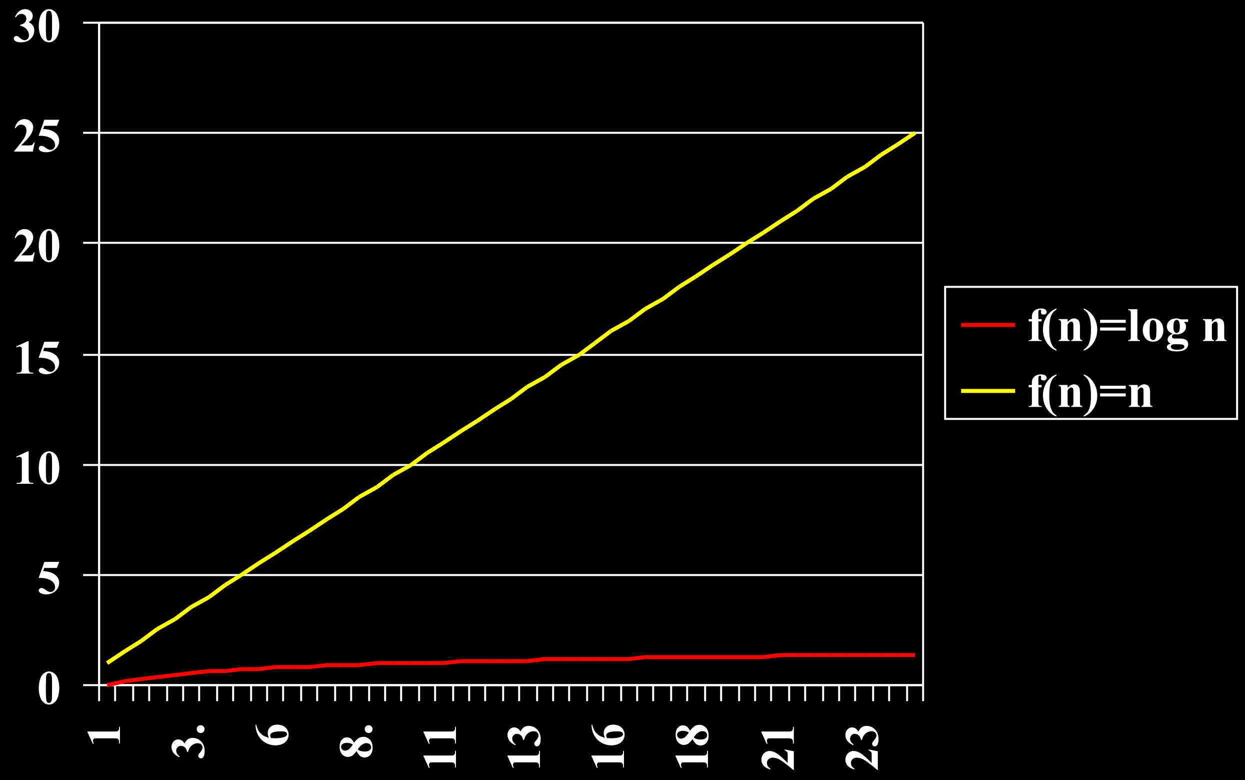
$$\lim_{n \to b} \frac{n^2}{e^n} = \lim_{n \to b} \frac{2\text{n}}{e^n} = \lim_{n \to b} \frac{2}{e^n} = \lim_{n \to b} \frac{0}{e^n} = 0$$

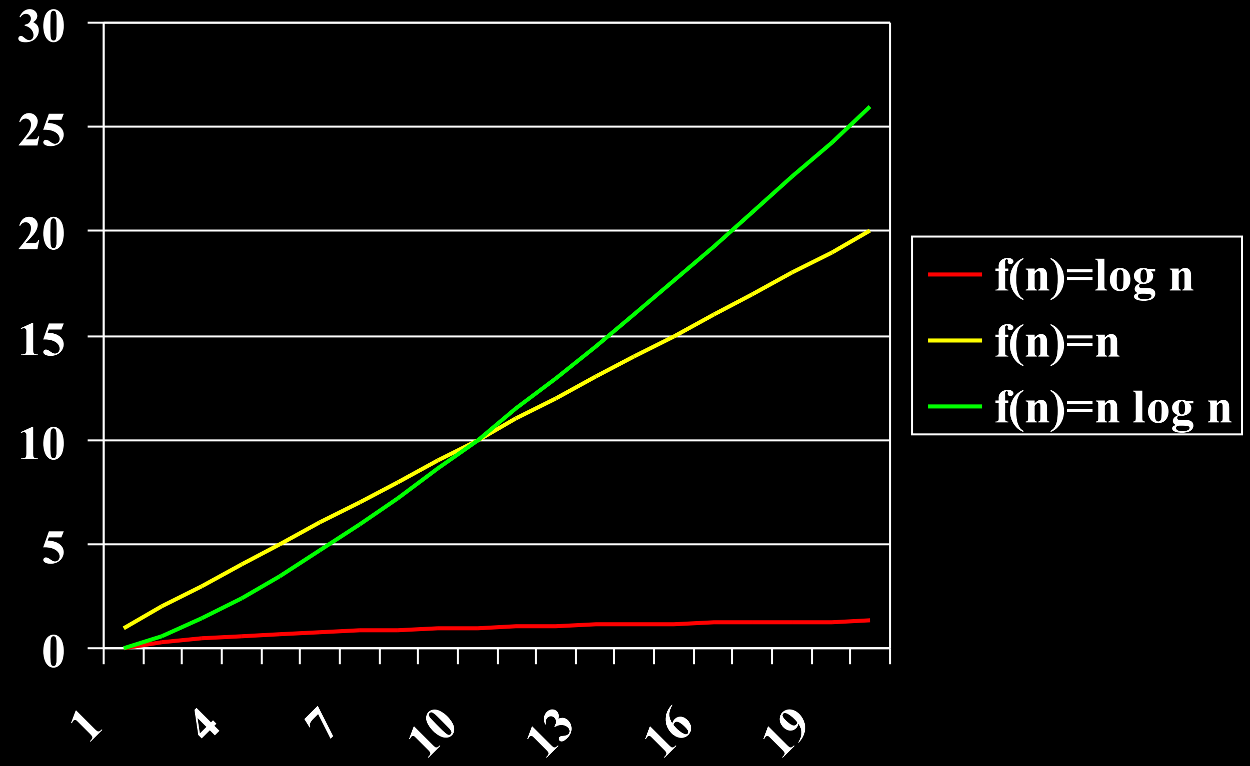
thus, $f_2(n)$ grows faster

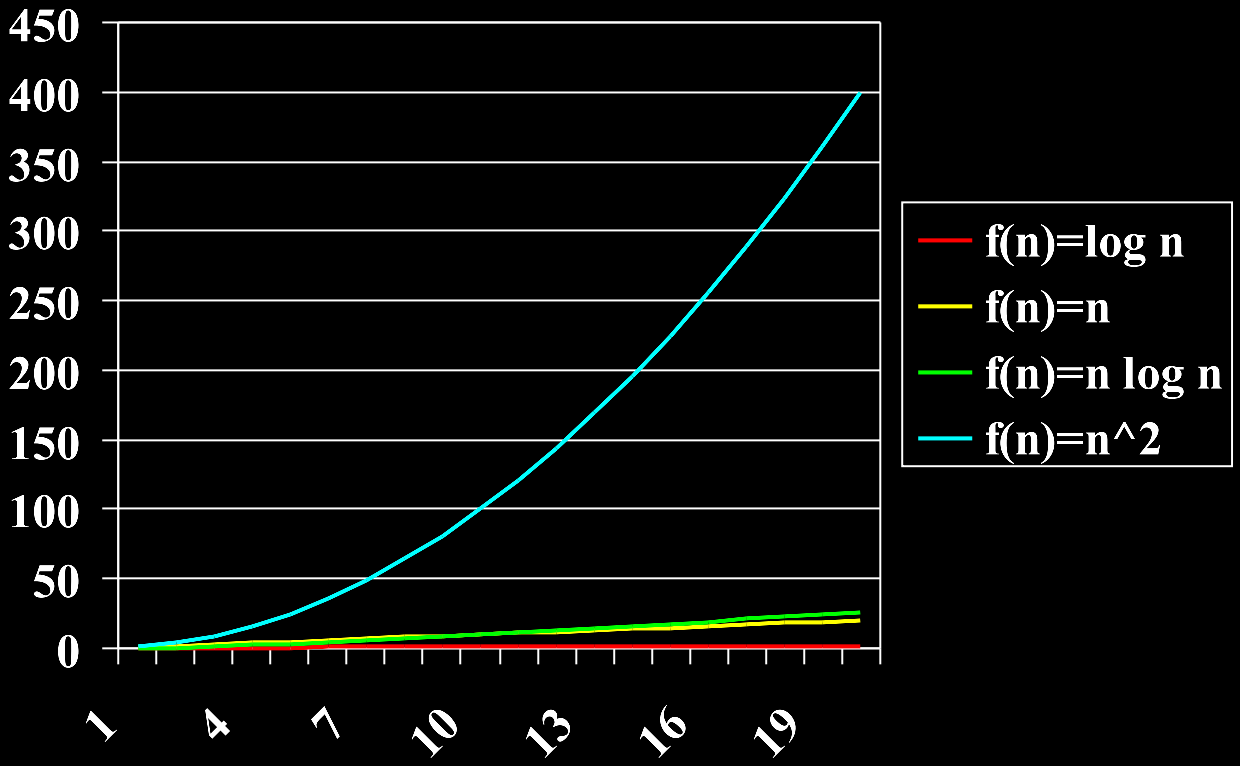


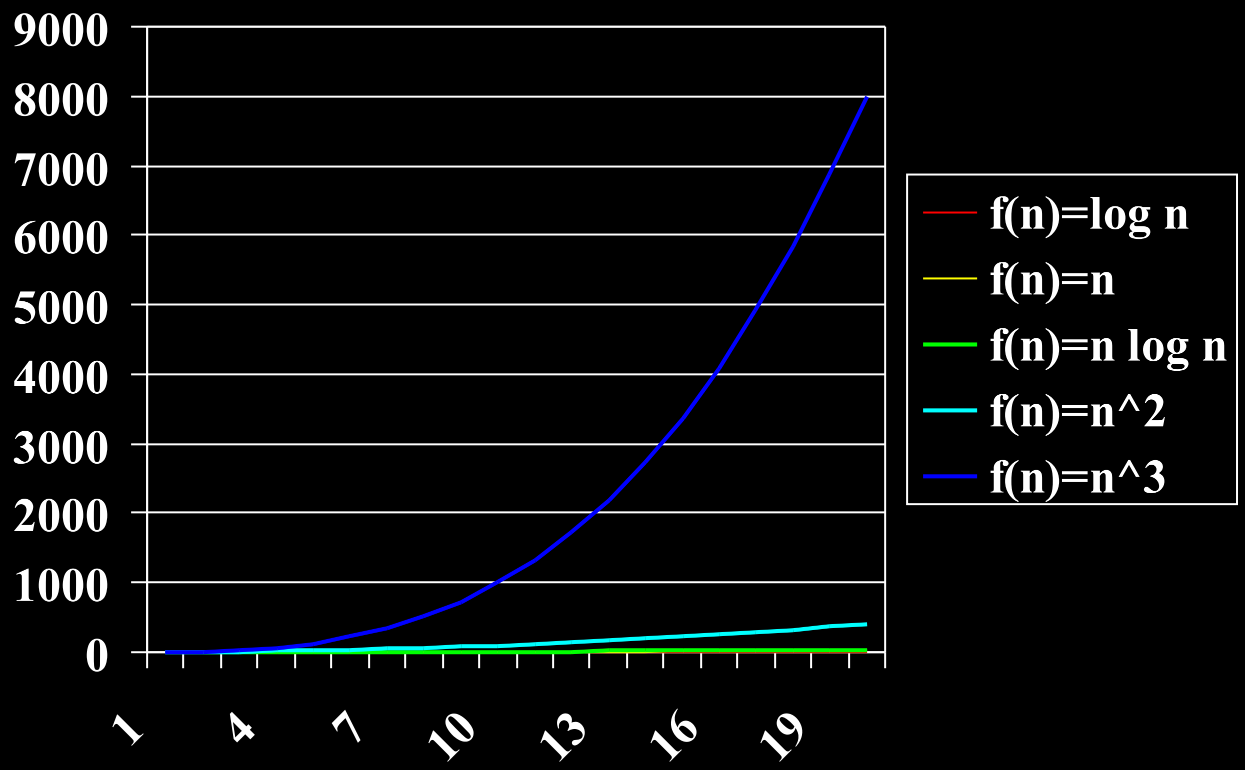


1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
3.
6
8.
11
13
16
18
21
23
f(n)=log n

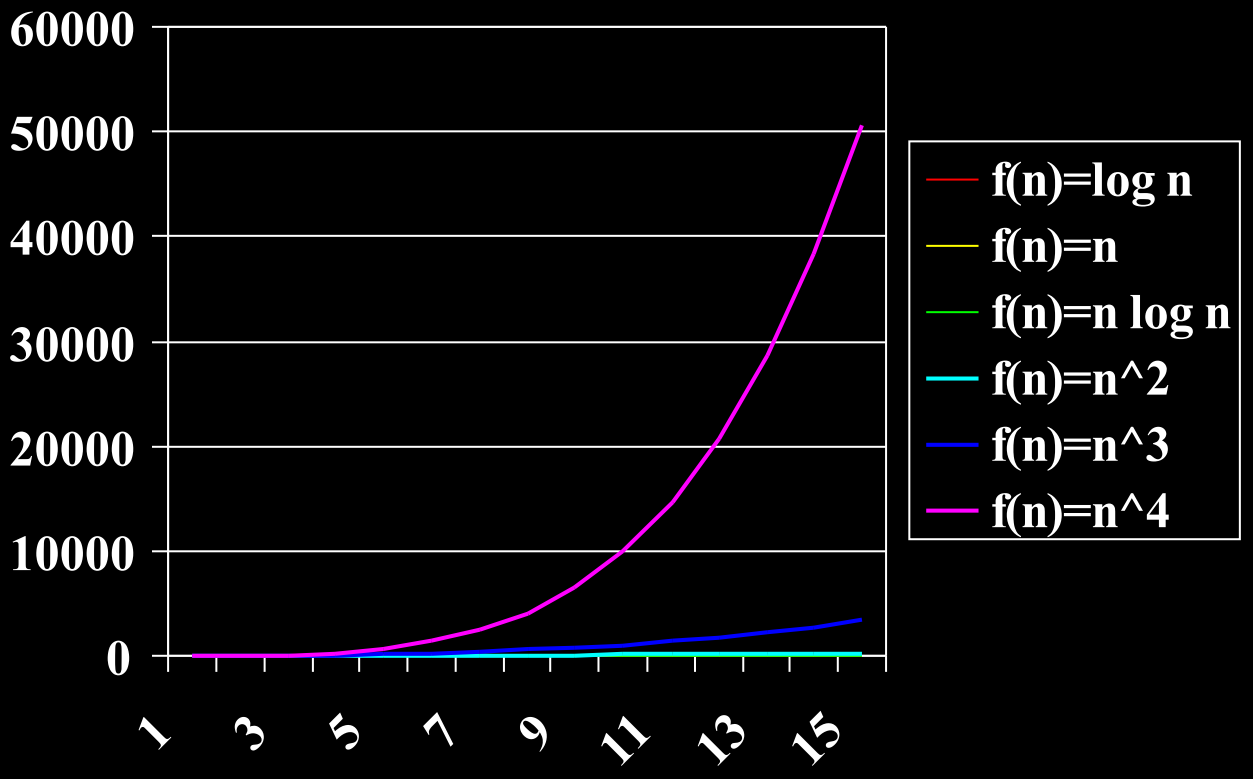

30
25
20
15
10
5
0
1
3.
6
8.
11
13
16
18
21
23
f(n)=log n
f(n)=n

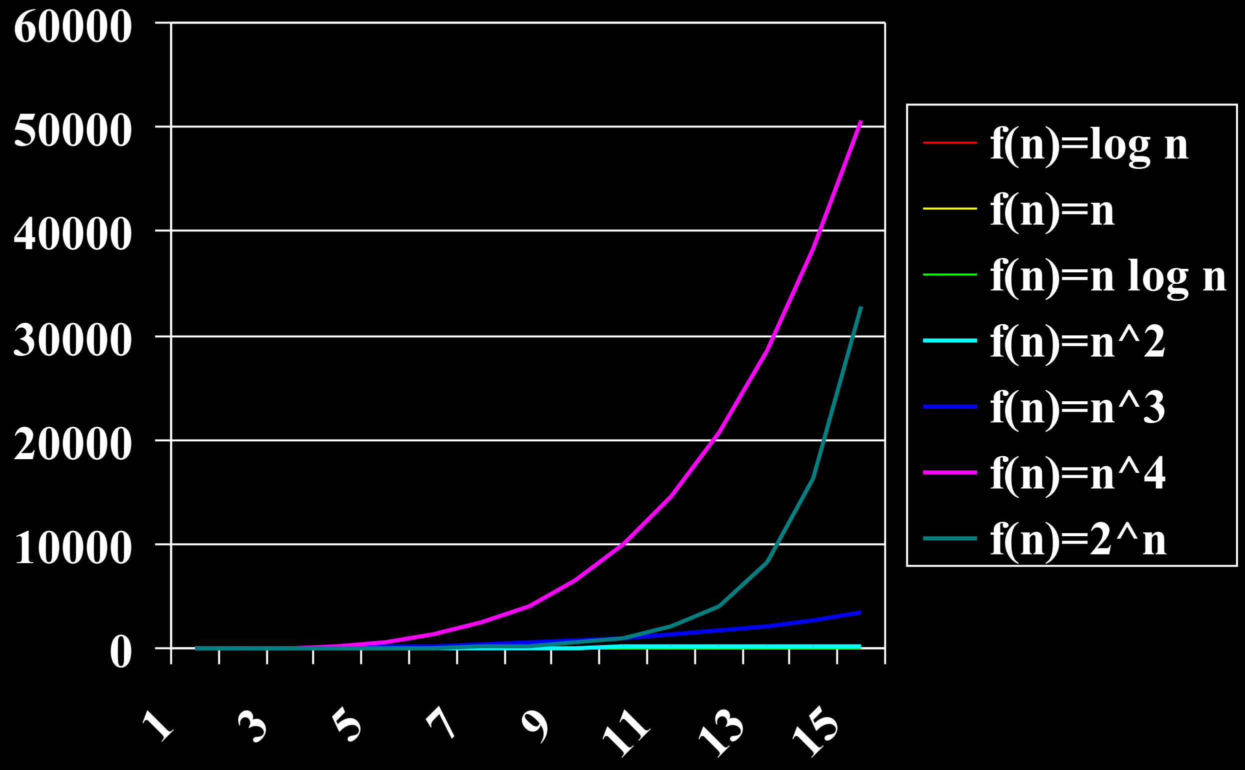

30
25
20
15
10
5
0
1
4
7
10
13
16
19
f(n)=log n
f(n)=n
f(n)=n log n

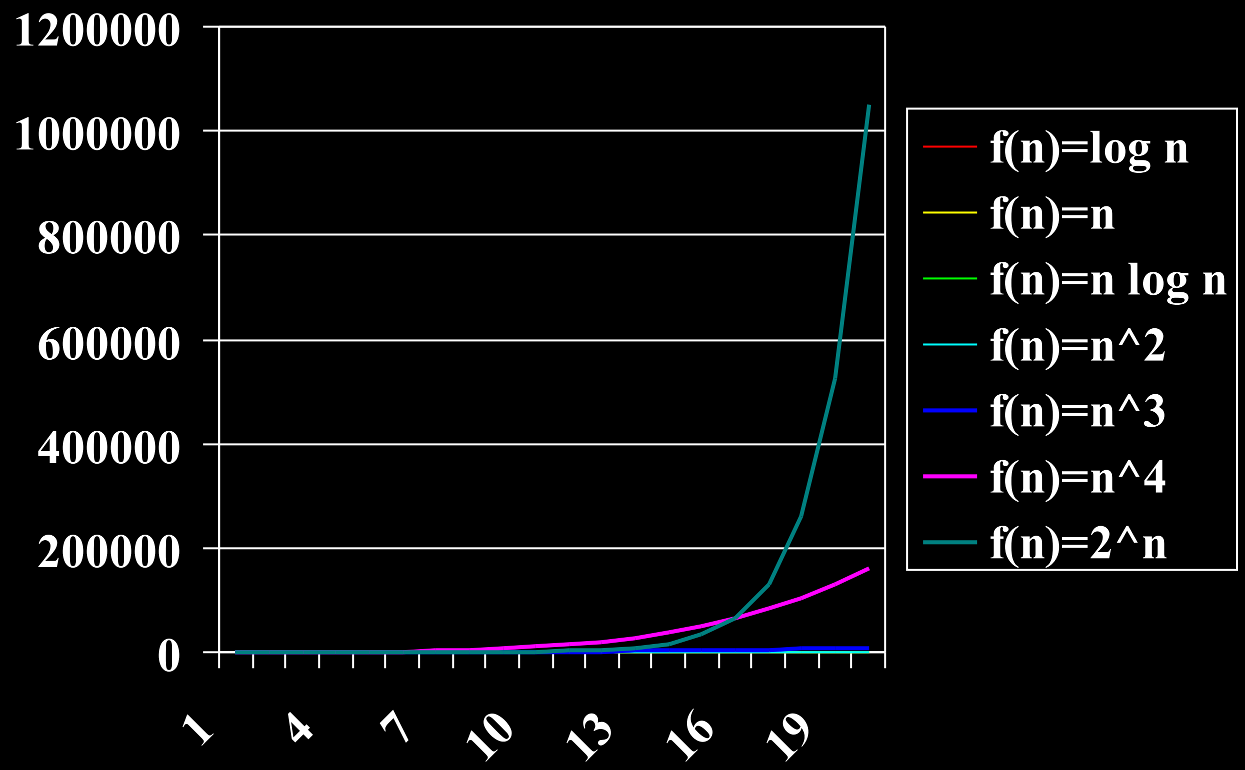

450
400
350
300
250
200
150
100
50
0
1
4
7
10
13
16
19
f(n)=log n
f(n)=n
f(n)=n log n
f(n)=n^2



9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1
4
7
10
13
16
19
f(n)=log n
f(n)=n
f(n)=n log n
f(n)=n^2
f(n)=n^3


60000
50000
40000
30000
20000
10000
0
1
3
5
7
9
11
13
15
f(n)=log n
f(n)=n
f(n)=n log n
f(n)=n^2
f(n)=n^3
f(n)=n^4


60000
50000
40000
30000
20000
10000
0
1
3
5
7
9
11
13
15
f(n)=log n
f(n)=n
f(n)=n log n
f(n)=n^2
f(n)=n^3
f(n)=n^4
f(n)=2^n



1200000
1000000
800000
600000
400000
200000
0
1
4
7
10
13
16
19
f(n)=log n
f(n)=n
f(n)=n log n
f(n)=n^2
f(n)=n^3
f(n)=n^4
f(n)=2^n

Functions Ordered by Growth and Rate

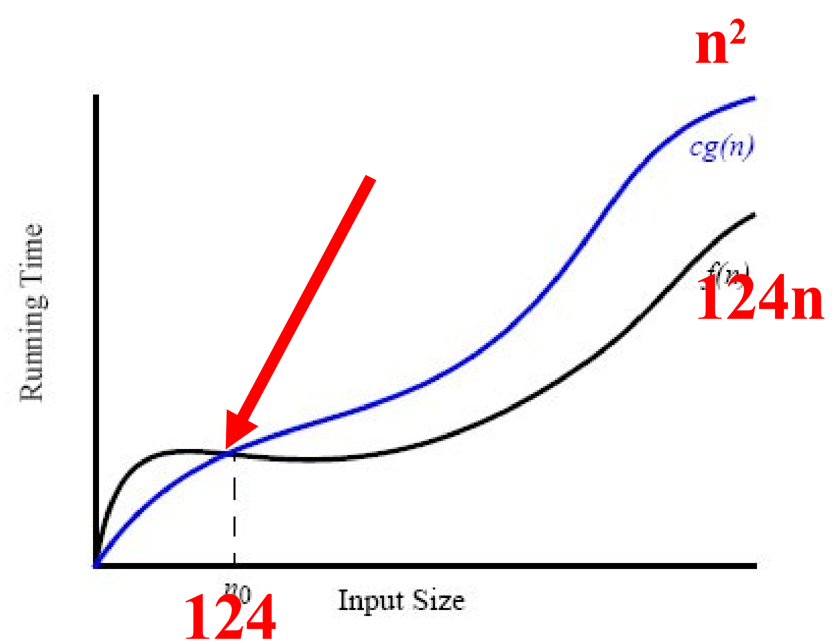
- $\log n$
- $\log^2 n$
- $\sqrt{n}$
- n
- $n \log n$
- n^2
- n^3
- 2^n

P = class of polynomial time algorithms

NP = class of *nondeterministic* polynomial time algorithms

Warning: O(n^2) can be "faster" than O(n) for small inputs

- $124n > n^2$ for $n = 1..123$
- $124n = n^2$ for $n = 124$
- $124n < n^2$ for $n > 124$



Big-Oh Rules

- If is $f(n)$ a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

Big-Omega Notation

Let f: IN→IR and g: IN→IR.

f(n) is Ω(g(n))

if and only if

g(n) is O(f(n))

IN: non-negative integers
IR: real numbers

Big-Theta Notation

Let f: $\mathbb{N} \rightarrow \mathbb{R}$ and g: $\mathbb{N} \rightarrow \mathbb{R}$.

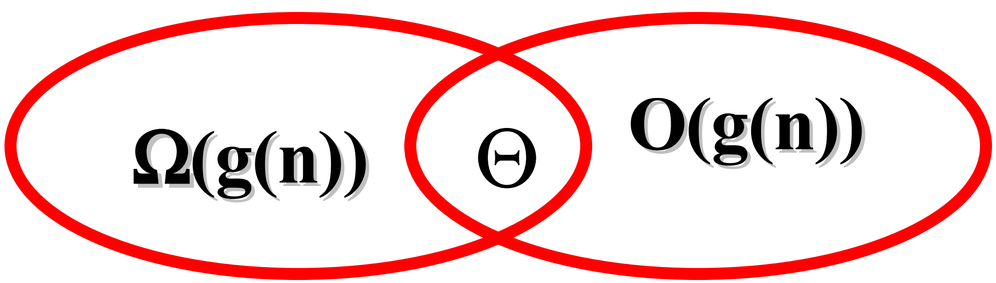
f(n) is $\Theta(g(n))$

if and only if

f(n) is $O(g(n))$ and f(n) is $\Omega(g(n))$.

Intuition of Asymptotic Terminology

- Big-Oh: $O(g(n))$ upper bound; functions that grow no faster than g(n)
- Big-Omega: $\Omega(g(n))$ lower bound; functions that grow at least as fast than g(n)
- Big-Theta: $\Theta(g(n))$ asymptotic equivalence; functions that grow at the same rate as g(n)



Little-Oh Notation

Let f: $\mathbb{N} \rightarrow \mathbb{R}$ and g: $\mathbb{N} \rightarrow \mathbb{R}$.

$f(n)$ is $o(g(n))$

if and only if

for **<u>any</u>** constant $c > 0$ there is a constant $n_0 > 0$ such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$.

Little-Omega Notation

Let f: IN→IR and g: IN→IR.

f(n) is ω(g(n))

if and only if

g(n) is o(f(n)).

Intuition of Asymptotic Terminology

- Big-Oh: upper bound
- Big-Omega: lower bound
- Big-Theta: asymptotic equivalence
- Little-Oh: less than (in asymptotic sense). The bound is not asymptotically tight.
- Little-Omega: greater than (in asymptotic sense). The bound is not asymptotically tight.

This lecture

- Asymptotic notations
  - why asymptotic notations
  - Big-Oh
    - $f(n) \le c \cdot g(n)$ for all $n \ge n_0$
  - Big-Omega, Big-Theta
    - little-oh, little-omega
- Explore further
  - $n!$ is $O(2^n)$?
  - 2^n is $O(n!)$?

Next lecture

- Case studies
  - read AD Chapter 1