CSC 225

Algorithms and Data Structures I
Fall 2014
Rich Little

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 $\Omega(.)$
- Big-Theta $\Theta(.)$
- Little-Oh $o(.)$
- Little-Omega $\omega(.)$

Big-Oh Notation

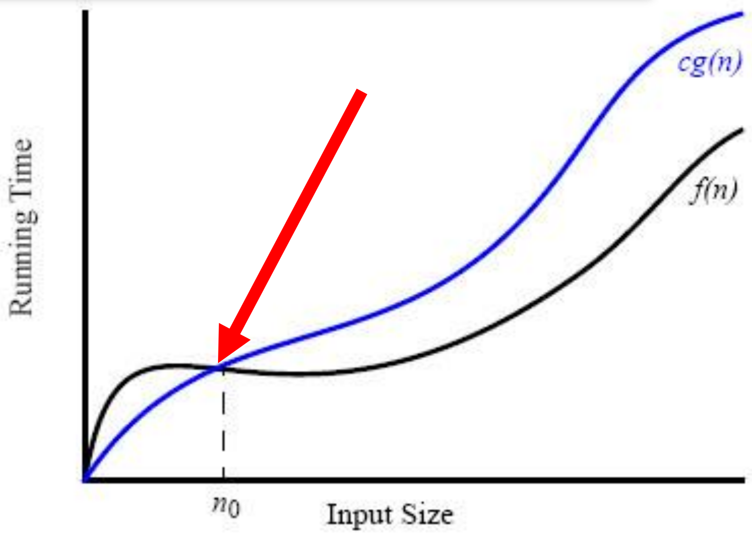
- Formal definition
- Most frequently used
- Is a good measure for *large* inputs

Formal Definition of Big-Oh Notation

Let $f: \mathbb{N} \to \mathbb{R}$ and $g: \mathbb{N} \to \mathbb{R}$. $f(n)$ is $O(g(n))$ if and only if
there exists a real constant $c > 0$
and an integer constant $n_0 > 0$
such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.
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: $4n + 20n^4 + 117 \leq 90n^4$
- $f(n) = 1083$
  O(f(n)) is **O(1)**
- $f(n) = 3 \log n$
  O(f(n)) is **O(log n)**
  P: $3 \log n \leq 4 \log n$
- $f(n) = 3\log n + \log \log n$
  O(f(n)) is **O(log n)**
  P: $3\log n + \log \log n \leq 4 \log n$
- $f(n) = 2^{17}$
  O(f(n)) is **O(1)**
  P: $2^{17} \leq 1\ 2^{17}$
- $f(n) = 33/n$
  O(f(n)) is **$O(1/n)$**
  P: $33/n \leq 33(1/n)$ for $n \geq 1$
- $f(n) = 2^{\log_2 n}$
  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

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 $f(n)$ is $O(nd)$
- **R6:** n^x is $O(an)$ 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 $y > 0$

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 $a > 1$

Quiz
Which statement is True?

1. 2^n is $O(n!)$?

2. $n!$ is $O(2^n)$?

Quiz: 2^n is $O(n!)$ is true

Proof. Let $n \geq 4$. Then $n! > 2^n$.

Therefore, for $n_0 = 4$, $c = 1$, and $n \geq n_0$

Prove the claim by induction.

Quiz: 2^n is $O(n!)$ is true

Induction on *n*. We need to show that

$$n! > 2^n \quad \text{for all } n \geq 4 \qquad \textbf{(hypothesis)}$$

Base case: $n = 4$

$$4! > 2^4 \Leftrightarrow \cancel{4}\cdot 3\cdot \cancel{2}\cdot 1 > \cancel{2}\cdot \cancel{2}\cdot \cancel{2}\cdot 2 \quad \textbf{ok } \mathbf{3 > 2}$$

$n \to n+1$: Show: $(n+1)! > 2^{n+1}$ for all $n \geq 4$

$$(n+1)! = (n+1)\, n! > (n+1)2^n > 2\cdot 2^n = 2^{n+1}$$

Hypothesis: $n! > 2^n$

$\Leftrightarrow$

$$(n+1)! > 2^{n+1}$$

$\boldsymbol{n+1 > 4}$

Stirling's Formula

- Another useful formula for ordering functions by growth rate is Stirling's Formula (1730)

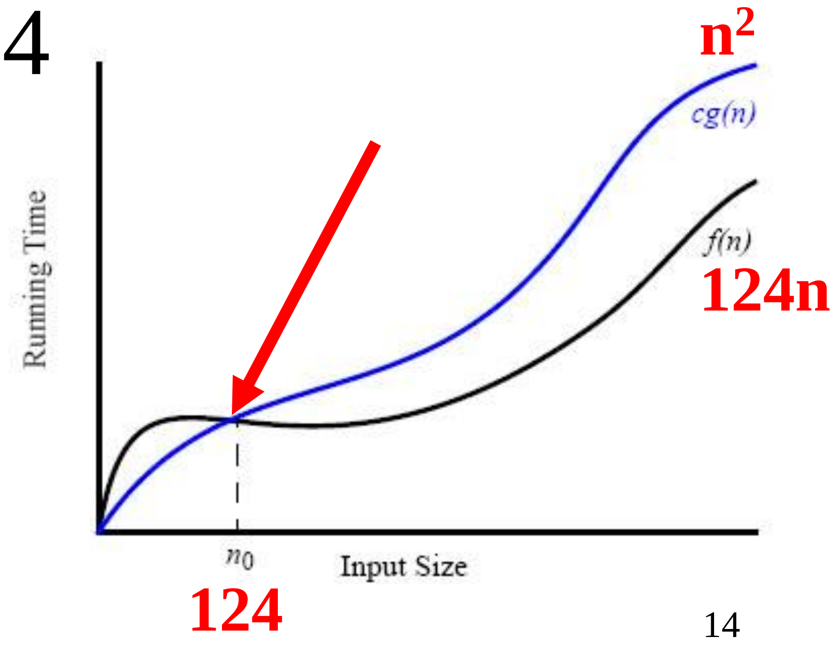
$$n! \approx \sqrt{2\pi n}\left[\frac{n}{e}\right]^n$$

Quiz

- Can an algorithm with running time O(n^2) be faster than an algorithm with running time O(n) for *small* inputs?

Quiz: 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 $n > 124$



Most Common Functions in Algorithm Analysis Ordered by Growth

$$1 \quad \log\log n \quad \log n \quad \sqrt{n}$$

$$n \quad n\log n \quad n^2 \quad n^{2.31}$$

$$n^3 \quad n^k, \text{for } k > 3$$

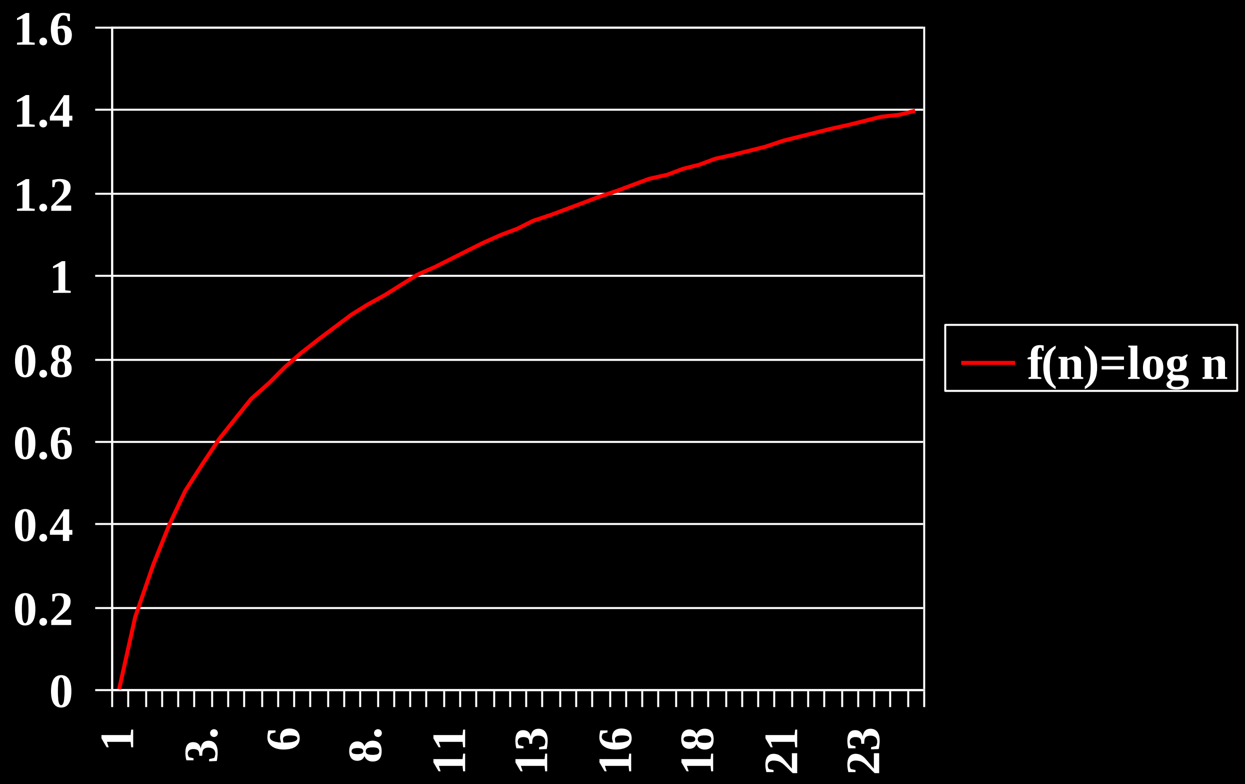
$$2^n \quad 3^n \quad n!$$

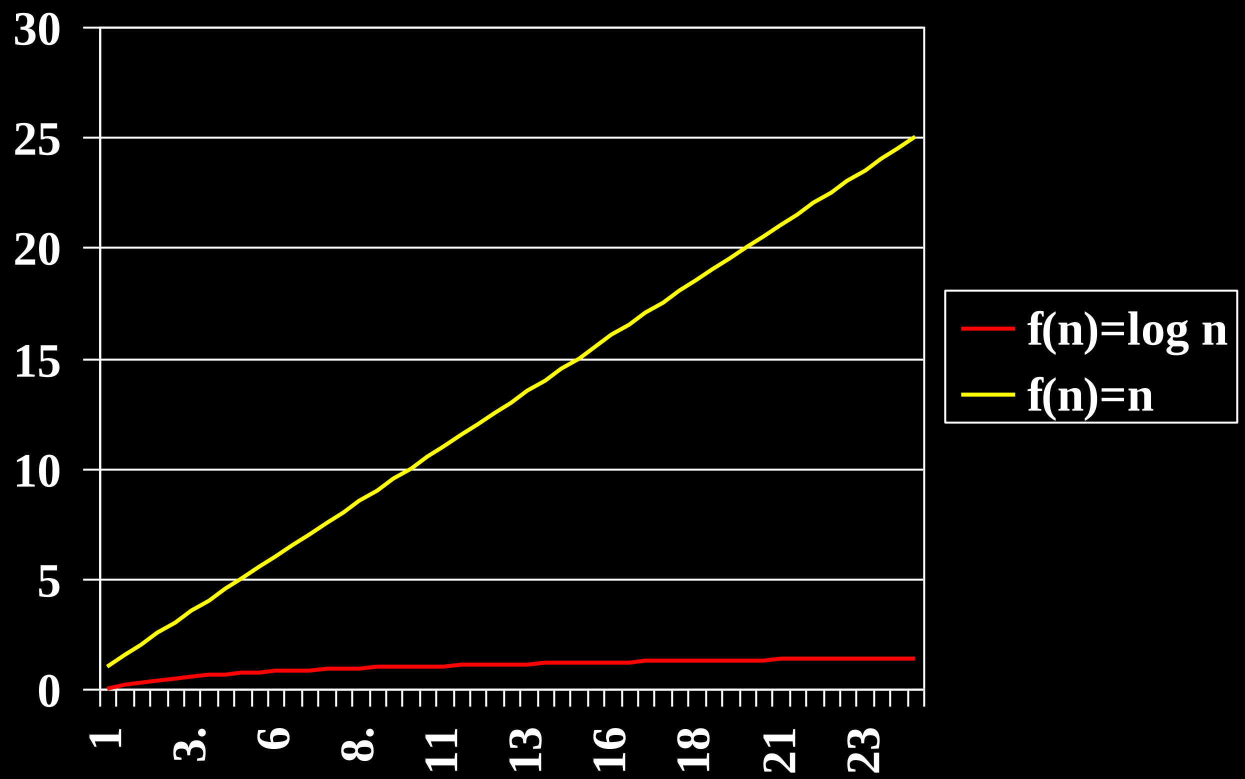
Examples of Most Common Big Oh Functions

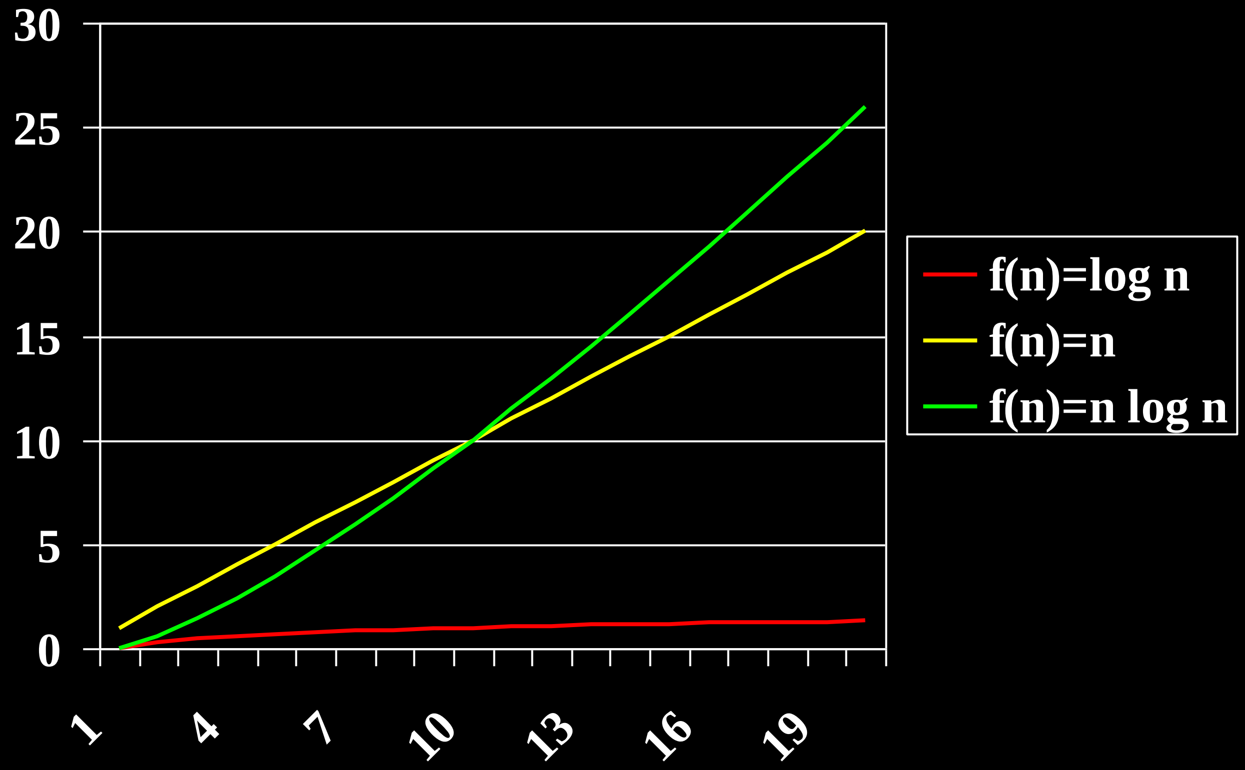
- Constant time $O(1)$
- Logarithmic $O(\log n)$
- Linear $O(n)$
- Quadratic $O(n^2)$
- Polynomial $O(n^k)$
- Exponential $O(a^n)$

Examples of Most Common Big Oh Functions

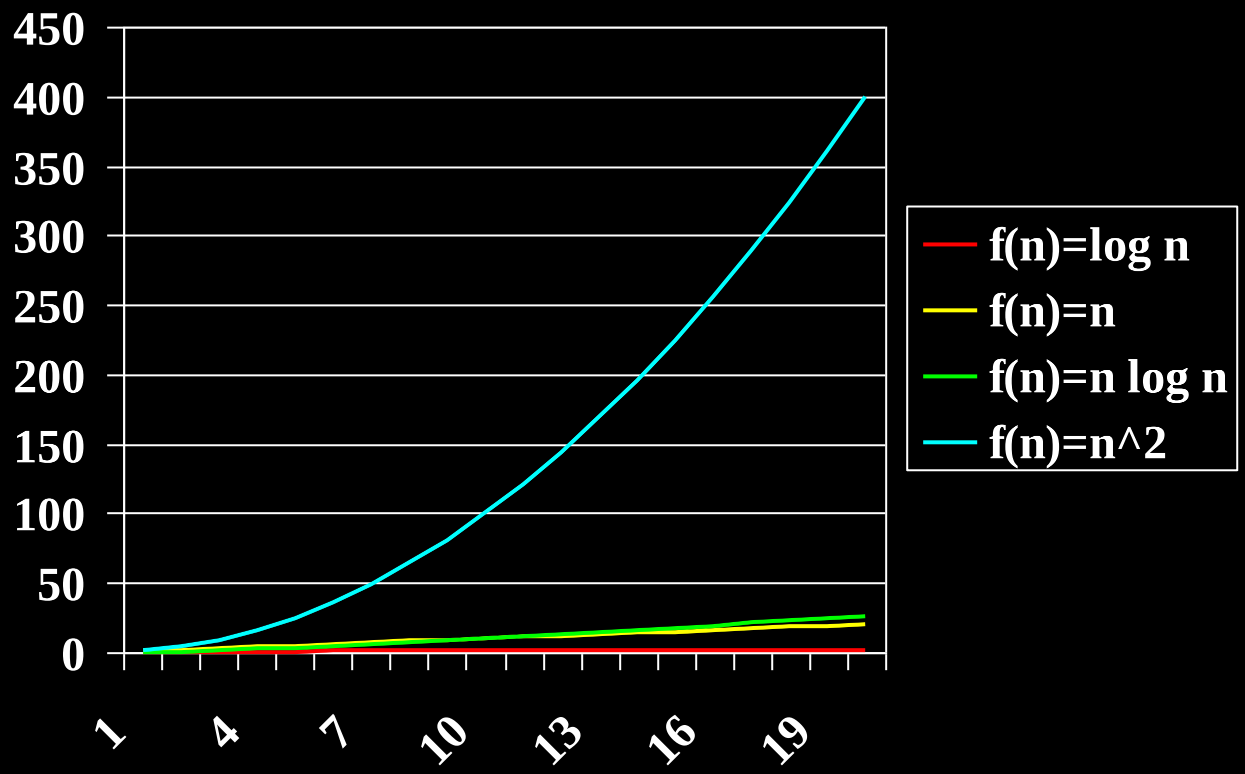
Constant	$O(1)$	Hash search
Logarithmic	$O(\log n)$	Tree search
Linear	$O(n)$	Linear search Linear median
	$O(n \log n)$	Heapsort
Quadratic	$O(n^2)$	Insertionsort
Cubic	$O(n^3)$	Transitive closure
Polynomial	$O(n^k)$	
Exponential	$O(2^n)$	Graph colouring
Exponential	$O(a^n)$	NP-hard problems

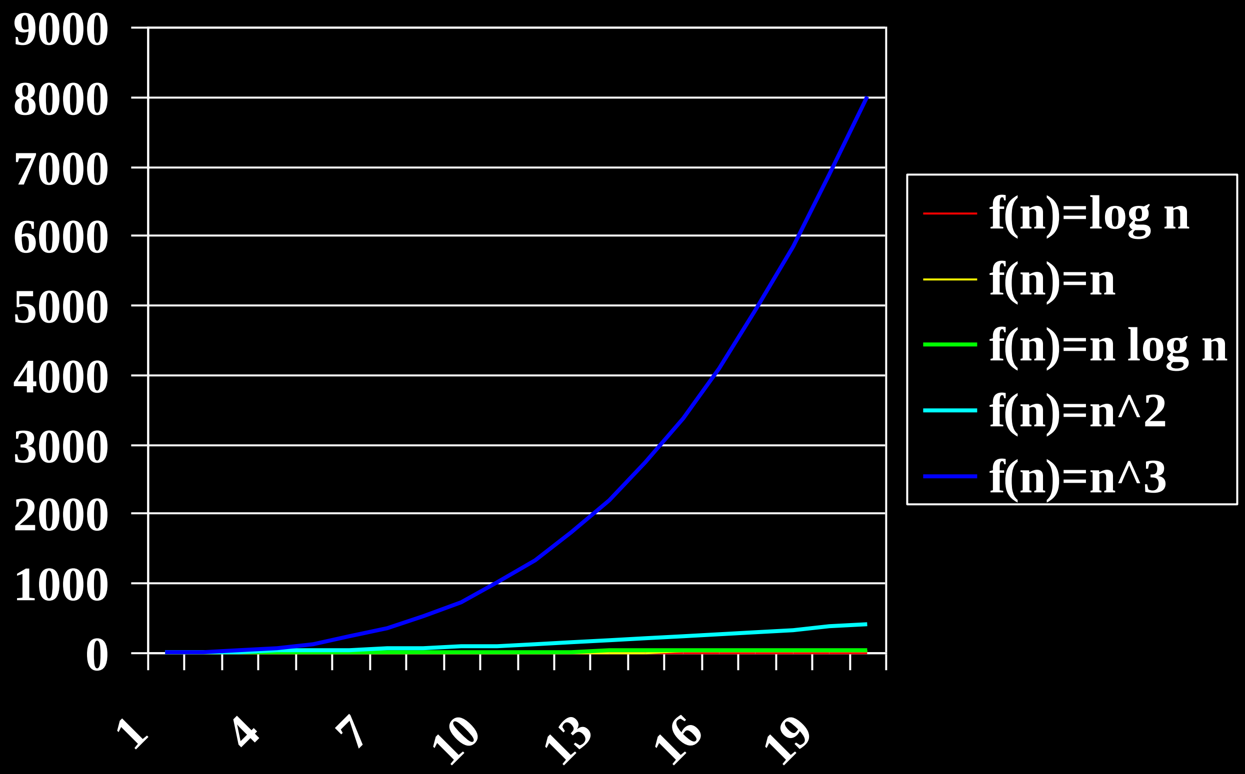

1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
3.
6
8.
11
13
16
18
21
23
f(n)=log n

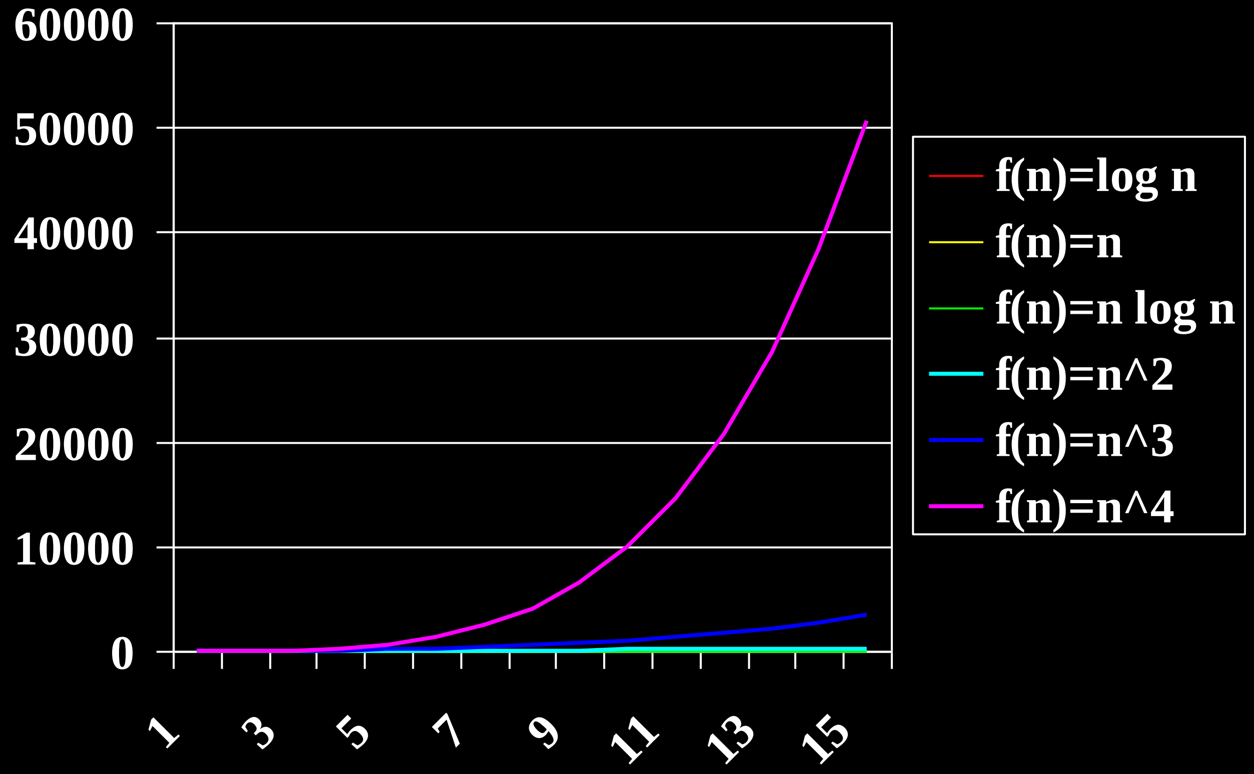

30
25
20
15
10
5
0
1
3.
6
8.
11
13
16
18
21
23
f(n)=log n
f(n)=n

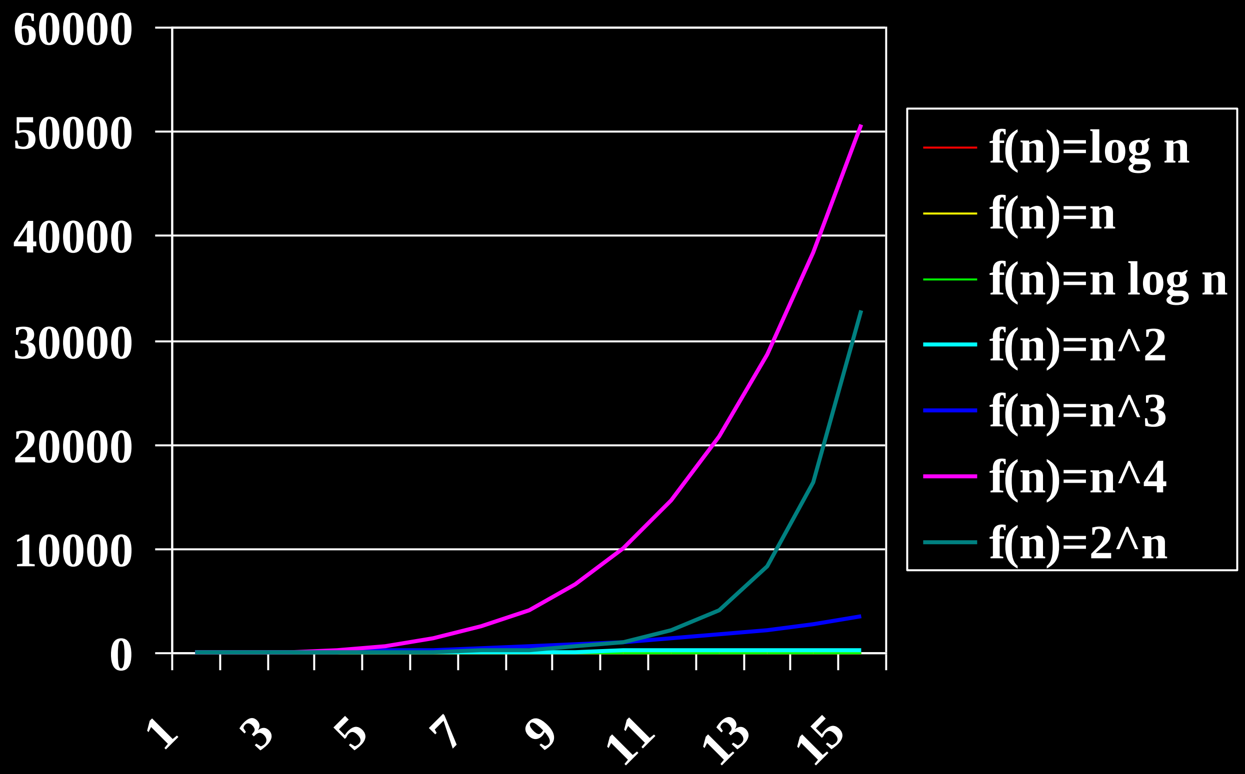


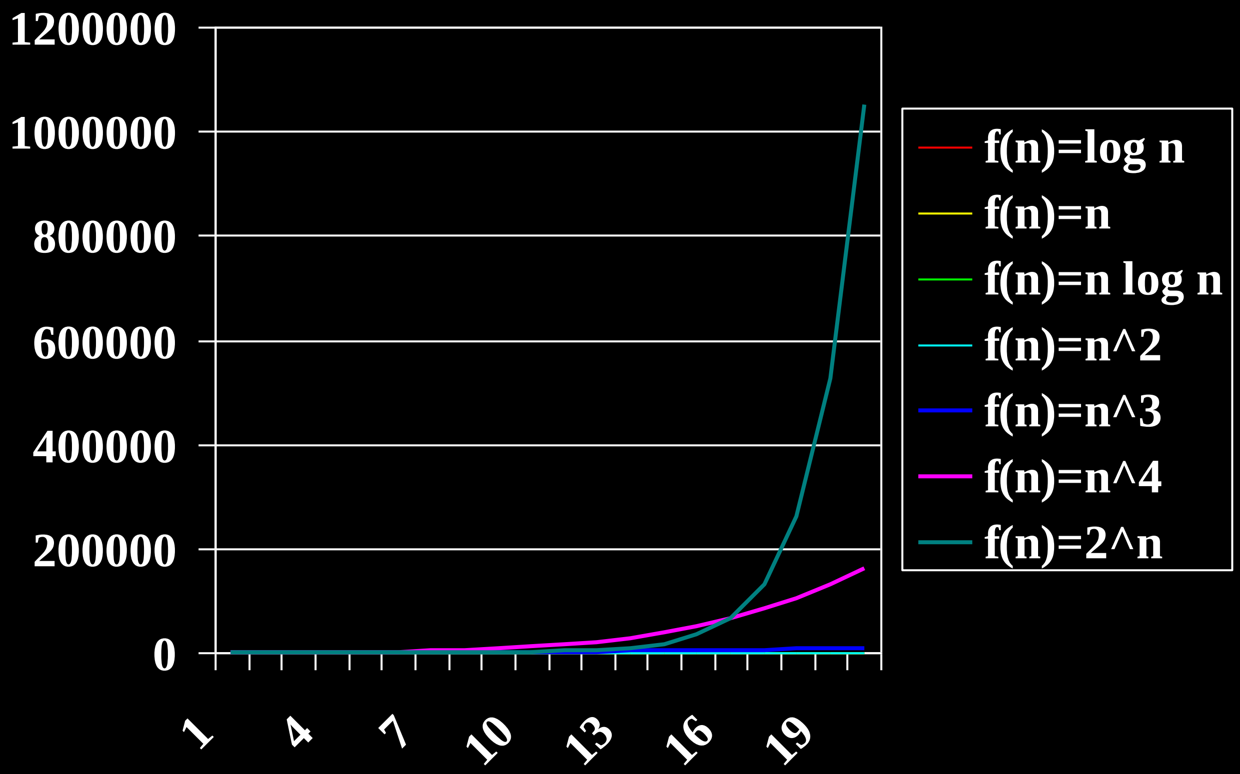
30
25
20
15
10
5
0
1
4
7
10
13
16
19
f(n)=log n
f(n)=n
f(n)=n log n


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

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 $x10^2$	10^4	10^6	10^{30}	10^{158}
10^3	10	10^3	10 $x10^3$	10^6	10^9		
10^4	13	10^4	13 $x10^4$	10^8	10^{12}		
10^5	17	10^5	17 $x10^5$	10^{10}	10^{15}		
10^6	20	10^6	20 $x10^6$	10^{12}	10^{18}		

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

Functions Ordered by Growth and Rate

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

P = class of polynomial time algorithms

- 2^n

NP = class of *nondeterministic* polynomial time algorithms

A million (US-) dollar question

http://www.claymath.org/millennium/P_vs_NP/

- P = NP?
- Obviously $P \subseteq NP$.
- There is a bunch of problems (so called NP-complete problems) for which one assumes that none of those can be solved in polynomial time.
- Examples: Graph coloring, Independent Set, Generalized 15-Puzzle
- Widely assumed: $P \neq NP$

Logarithms and Exponential Functions

- Review properties of Logarithms and exponents

$$\log_b a = c \text{ if } \; a = b^c$$

$$\log ac = \log a + \log c$$

$$\log a / c = \log a - \log c$$

$$\log a^c = c \log a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$b^{\log_c a} = a^{\log_c b}$$

$$(b^a)^c = b^{ac}$$

$$b^a b^c = b^{a+c}$$

$$b^a / b^c = b^{a-c}$$

Useful Summations Formulas

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} \quad \text{(Gauss)}$$

$$\sum_{k=1}^{n} k^2 = \frac{2n^3+3n^2+n}{6} = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \left[\frac{n(n+1)}{2}\right]^2$$

$$\int_{a-1}^{b} f(x)dx \le \sum_{k=a}^{b} f(k) \le \int_{a}^{b+1} f(x)dx$$

Big-Omega Notation

Let f: IN$\rightarrow$IR and g: IN$\rightarrow$IR.

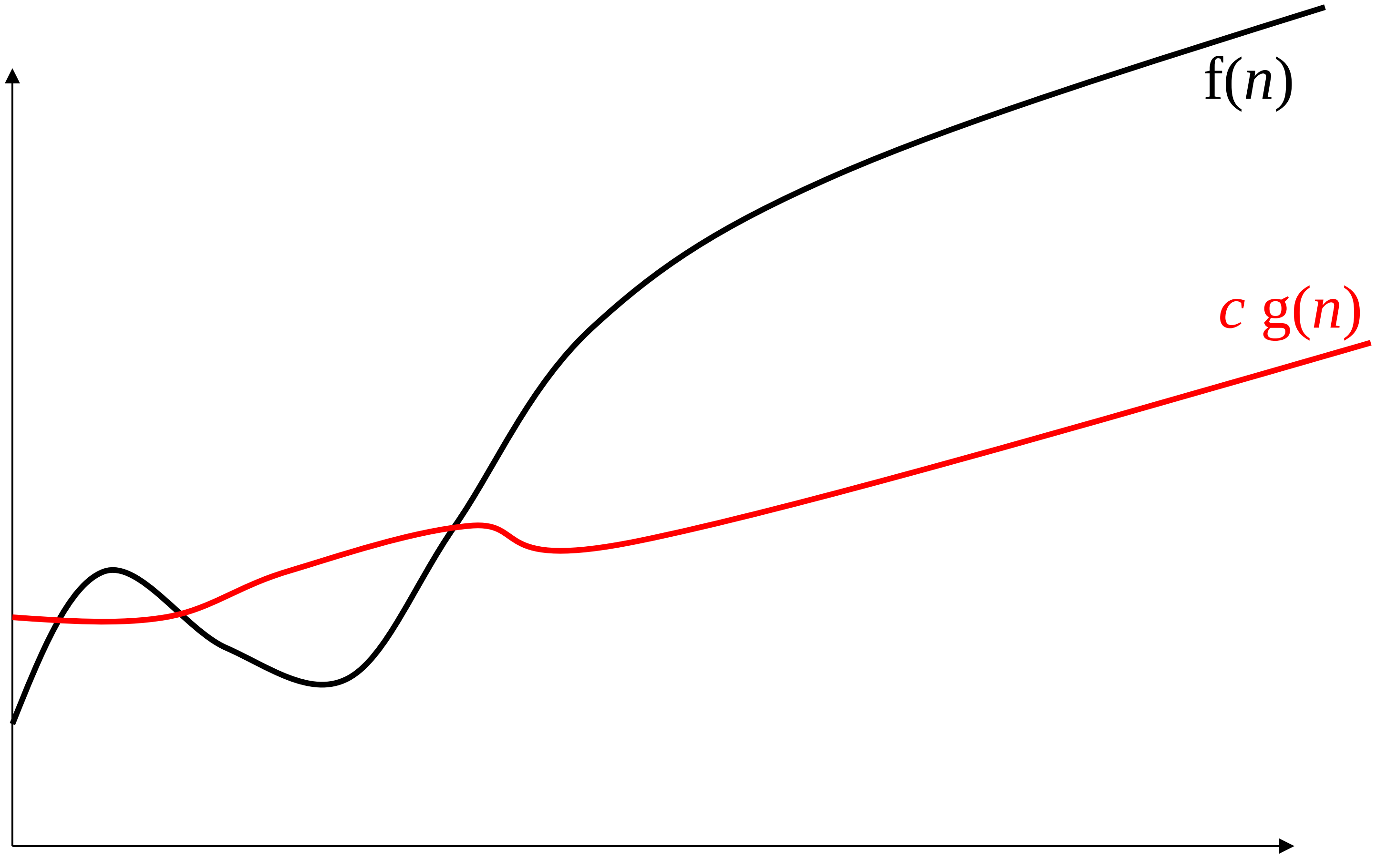
$f(n)$ is $\Omega(g(n))$

if and only if

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

IN: non-negative integers
IR: real numbers

f(n) = Ω(g(n))



Quiz: What is true, what is false?

f(n) is Ω(g(n))
iff
g(n) is O(f(n))

1. 2^n is $\Omega(n!)$

Previous results
2^n is $O(n!)$ is true
$n!$ is not $O(2^n)$

2. $n!$ is $\Omega(2^n)$

2^n is $\Omega(n!)$ is false

$f(n) = 2^n$
$g(n) = n!$

$f(n)$ is $\Omega(g(n))$
iff
$g(n)$ is $O(f(n))$

We know 2^n is $O(n!)$ but $n!$ is not $O(2^n)$.

Since 2^n is $\Omega(n!)$ iff $n!$ is $O(2^n)$, the claim is false.

$n!$ is $\Omega(2^n)$ is true

f(n) = $n!$
g(n) = 2^n

f(n) is Ω(g(n))
iff
g(n) is O(f(n))

We know 2^n is $O(n!)$ but $n!$ is not $O(2^n)$.

Since $n!$ is $\Omega(2^n)$ iff 2^n is $O(n!)$, the claim is true.

Big-Theta Notation

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

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

if and only if

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

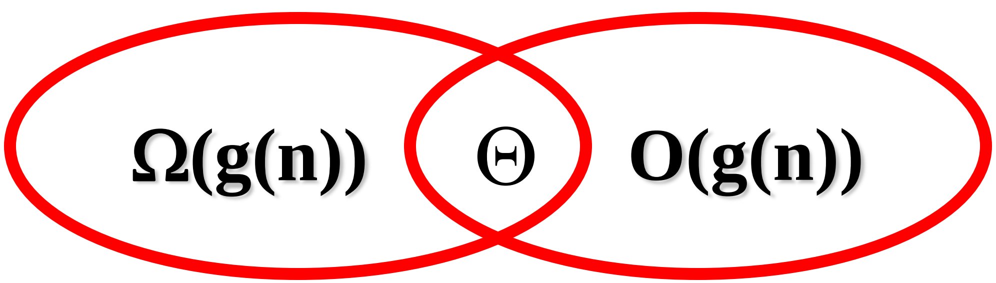
Big-Theta: Examples

- $3n + 1$ is $\Theta(n)$
- $2n^2 + 3n + 1$ is $\Theta(n^2)$

Review the examples for Big-Oh notation and solve those for big-Omega, big-Theta!

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)



Asymptotic Notation

- Big-Oh O(.)
- Big-Omega Ω(.)
- Big-Theta Θ(.)

- **Little-Oh o(.)**
- **Little-Omega ω(.)**

Little-Oh Notation

Let f: IN→IR and g: IN→IR.

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

if and only if

for any constant $c > 0$ there is a constant $n_0 > 0$ such that $f(n) \le c \cdot g(n)$ for $n \ge n_0$.

Examples: Little-Oh

- $2n$ is $o(n^2)$
- $2n^2$ is ***not*** $o(n^2)$! [but $2n^2$ is $O(n^2)$]

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 Notation

Let f: IN→IR and g: IN→IR.

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

if and only if

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

Little-Omega

- $2n^2$ is $\omega(n)$
- If $f(n)$ is $\omega(g(n))$ then $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$.

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.

Find an Algorithm to solve Prefix Averages Problem

- Note: Efficiency and Design go hand in hand!

Prefix Averages

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[k]$ is the average of elements $X[0], \ldots, X[k]$

X	12	3	7	24	4	1	1
A	12	7.5	7.3	11.5	10	8.5	7.4

Algorithm PrefixAverages

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[k]$ is the average of elements $X[0]$, …, $X[k]$

```
Let A be an array of n numbers.
for k←0 to n-1 do
   a←0
   for j←0 to k do
      a←a+X[j]
   end
   A[k]←a/(k+1)
end
return A
```

$k + 1$ times

$1 + 2 + 3 + \ldots + n = ?$

Worst-Case Running Time of Algorithm PrefixAverages

$$1+2+3+\ldots+n=$$

$$\sum_{i=1}^{n} k = \frac{1}{2} n(n+1) \text{ is } O(n^2)$$

Quiz: Can we solve the problem faster?

Prefix Averages

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[k]$ is the average of elements $X[0], \ldots, X[k]$

Quiz

- $A[0] = ?$
- $A[1] = ?$
- $A[2] = ?$

$\vdots$

- $A[k\text{-}1] = ?$
- $A[k] = ?$

Quiz

- $A[0] = X[0]$
- $A[1] = ?$
- $A[2] = ?$

$\vdots$

- $A[k\text{-}1] = ?$
- $A[k] = ?$

Quiz

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = ?$

- $A[k\text{-}1] = ?$
- $A[k] = ?$

Quiz

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = (X[0] + X[1] + X[2])/3$

$\vdots$

- $A[k\text{-}1] = ?$
- $A[k] = ?$

Quiz

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = (X[0] + X[1] + X[2])/3$

$\vdots$

- $A[k\text{-}1] = (X[0] + X[1] + \ldots + X[k\text{-}1])/k$
- $A[k] = (X[0] + X[1] + \ldots + X[k\text{-}1] + X[k])/(k+1)$

Quiz

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = (X[0] + X[1] + X[2])/3$

$\vdots$

- $A[k\text{-}1] = (X[0] + X[1] + \ldots + X[k\text{-}1])/\mathrm{k}$
- $A[k] = (X[0] + X[1] + \ldots + X[k\text{-}1] + \mathrm{X}[k])/(k+1)$

Quiz

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = (X[0] + X[1] + X[2])/3$

$\vdots$

- $A[k\text{-}1] = (X[0] + X[1] + \ldots + X[k\text{-}1])/\mathrm{k}$
- $A[k] = (X[0] + X[1] + \ldots + X[k\text{-}1] + \mathrm{X}[k])/(k+1)$

Quiz

Idea: Use part of the computation for $A[k\text{-}1]$ when computing $A[k]$!

- $A[0] = X[0]$
- $A[1] = (X[0] + X[1])/2$
- $A[2] = (X[0] + X[1] + X[2])/3$

⋮

- $A[k\text{-}1] = (X[0] + X[1] + \ldots + X[k\text{-}1])/k$
- $A[k] = (X[0] + X[1] + \ldots + X[k\text{-}1] + X[k])/(k+1)$

Algorithm PrefixAverages2

Input: An n-element array X of numbers.

Output: An n-element array A of numbers such that $A[k]$ is the average of elements $X[0], \ldots, X[k]$

```
Let A be an array of n numbers.
s←0
for k←0 to n−1 do
  s←s+X[k]
  A[k]← s/(k+1)
end
return A
```

n times

O(n)

PrefixAverages vs. PrefixAverages2

- PrefixAverages runs in *quadratic* time $O(n^2)$
- PrefixAverages2 runs in *linear* time $O(n)$

- Thus, PrefixAverages2 is more efficient!
- The analysis drove the design of PrefixAverages2 to a certain extent