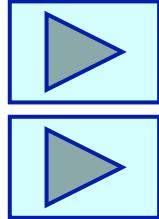
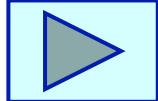


“Asymptotics, Summation in Analysis of Algorithms”

□ Lecture Topics and Readings



- ❖ Simple Analysis of Algorithms (AA-P) [CLRS]-C2
- ❖ Asymptotic Notations O , Θ , Ω , o , ω , [CLRS]-C3.1
- ❖ Summation and Bounding  [CLRS]-App.A

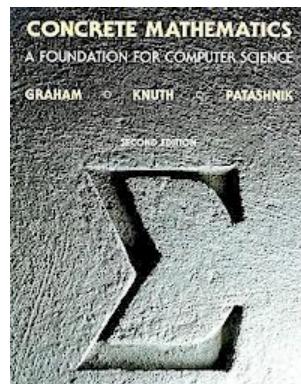
□ Assignment: T1 to start next week.

*Algorithms & Discrete Math
have very happy, stable marriage!*

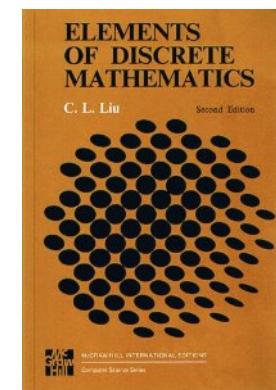
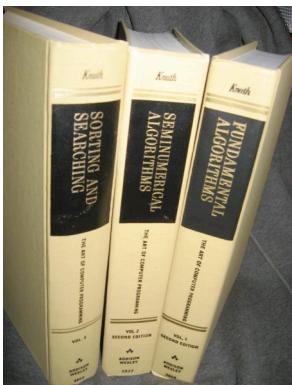
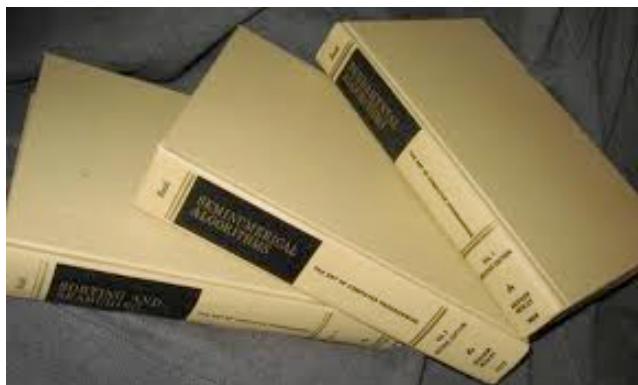
Early pioneers of Math for CS



Don Knuth
Stanford



C. L. Liu,
MIT, UIUC,
NTHU (tw)



Visiting Dave in Taipei (Feb 2006)



Thank you.

Q & A



School *of* Computing

CS3230 Algorithm Analysis

“Math for AA”

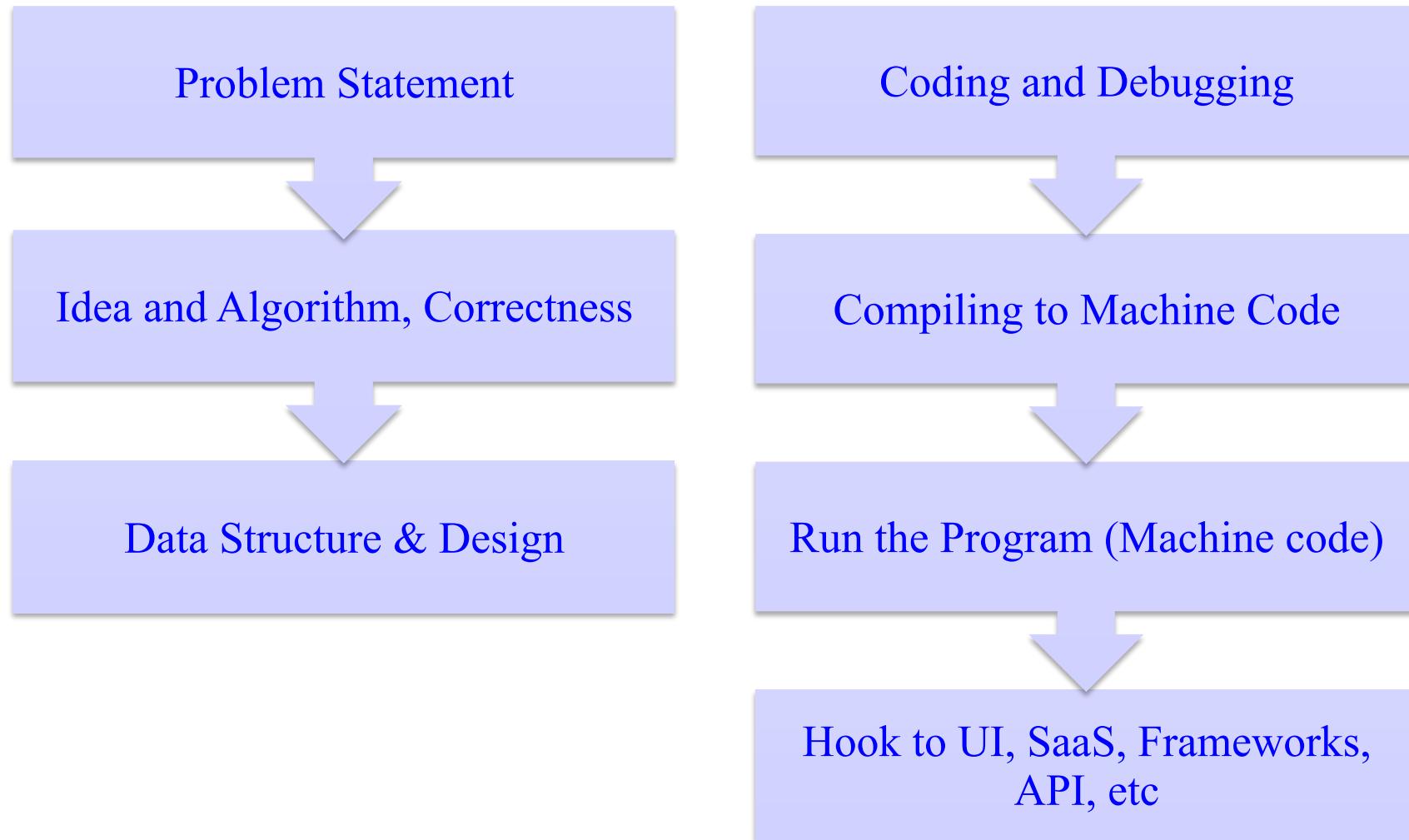
□ Lecture Topics and Readings

- ❖ Simple Algorithm Analysis [CLRS]-Ch2, CS2010
- ❖ Algorithm Analysis Pattern [LHW] A&E
- ❖ Asymptotic Notations [CLRS]-C3
- ❖ Functions & Summations (Review) [CLRS]-C3, App

Quickly get up to speed.



Problem to Algorithm to Code



Why not measure actual run-time?

What affects the run time of the *same* algorithm?

- * slightly different implementation code,
- * programming language (Java, C++, python, ...),
- * compiler used (& compiler options),
- * processor speed of computer,
- * work load on the system,

We cannot just measure *actual run-time*

- * not accurate (too many complicated factors)
- * not very conclusive

We need a model for analysis...

RAM Model of Computation

RAM = Random Access Model

- * Can access any memory in $O(1)$ time,
- * Sequential execution, no concurrent operations
- * Each memory stores integer or floating point
- * Perform basic arithmetic/logic ops in $O(1)$ time,
(Similar to real computers. See [CLRS]-Ch2.2)

Consider algorithm implemented in RAM model.
Then *running time* of algorithm is the *number of primitive RAM operations/instructions executed*.

Analysis using the RAM Model?

Same algorithm, but machine code has
slightly different (# of RAM instructions)

Example:

or just instructions

- Code 1: $(3n^2 + 17n)$ instructions (RAM model)
- Code 2: $(3n^2 + 17n)$ instructions (RAM model)

What is preserved? (...remain invariant)?

Both have *same rate-of-growth*

Both are $\Theta(n^2)$

Analysis of Algorithm

We do not measure *actual run-time*.

We measure running time as
“number of RAM instructions executed”

In practice, we

- * estimate # of instructions executed,
- * approximate using O , Ω , Θ notations
- * want to know *rate-of-growth* of running time



Trivial Algorithm Analysis



PASS-FAIL (*Mark*) ▷

“pseudocode”

Print Mark

if *Mark* < 40

then *print* “Student Fails”

else *print* “Student Passes”

AA. Simple. 3-4 elementary instructions.

Constant time.

(*always* take the same amount of time)

Does not vary with problem size n .

Trivial Algorithm Analysis

MPG ()

Read StartMiles, EndMiles, GasUsed

Distance \leftarrow (*EndMiles* – *StartMiles*)

Average \leftarrow *Distance* / *GasUsed*)

Print “Average mileage is “, *Average*

if *Average* ≥ 25

then Print “ *Mileage is good* ”

then Print “ *Mileage is not good* ”

Print “*Bye Bye* ”

“pseudo-code”

- AA.** Simple. About 10 instructions.
Constant time. (Does not vary with n .)

Trivial Algorithm Analysis



```
TEST ( $A, n$ )     $\triangleright A[1 \dots n]$       “pseudo-code”  
    Print  $A[1]$   
     $key \leftarrow A[57] * 2$   
    if  $A[1] > 0$  then Print “first number is positive”  
    Print  $A[n]$  (* print last element of array *)
```

AA: Simple. 4 instructions. Constant time.
Does not vary with n .

$$T(n) \geq 1, \quad T(n) \leq 1000, \quad T(n) = \Theta(1)$$

Lower bound

Upper bound

Guaranteed
running time



Trivial Algorithm Analysis



ARRAY-SUM (A, N) $\triangleright A[1 \dots N]$

$Sum-sf \leftarrow 0$

for $j \leftarrow 1$ **to** N

do $Sum-sf \leftarrow Sum-sf + A[j]$

$Sum \leftarrow Sum-sf$

“pseudo-code”

AA: Simple. About $(2n+2)$ instructions.
Linear time.
Directly proportional to n .

$$T(n) \geq n, \quad T(n) \leq 10,000n, \quad T(n) = \Theta(n)$$

Lower bound

Upper bound

Guaranteed
running time

(CS3230 Algorithm Analysis) Page 12

Trivial Algorithm Analysis



```
ALL-PRODUCT ( $A, n, M, n$ )  $\triangleright A[1..n], M[1..n; 1..n]$   
(* compute all products *)  
for  $k \leftarrow 1$  to  $n$                             “pseudo-code”  
    for  $j \leftarrow 1$  to  $n$   
        do  $M[k,j] \leftarrow A[k] * A[j]$ 
```

AA: About $(2n^2 + n)$ instructions
Quadratic time.
Proportional to n^2 .

$$T(n) \geq n^2, \quad T(n) \leq 10,000n^2, \quad T(n) = \Theta(n^2)$$

Lower bound

Upper bound

Guaranteed
running time



Testing * operation in a CPU

Q: How to test that the “*” operation of your CPU is correct?

A: Check exhaustively. for all a, b

$$\text{Check } a * b = c$$



Q: How long will it take?

A: Any guesses?



Testing * operation in a CPU

Q: How long will it take?

Assume we use a 100G-Flop CPU
can take 100B operations per sec.

a is a 32-bit number (2^{32} cases)

b is a 32-bit number (2^{32} cases)

So, $(a * b)$ there are (2^{64} cases)

Time taken = $(2^{64} / 100 \times 10^9)$ sec

≈ 6 years!





2⁶⁴ / 100x10⁹ seconds



≡ Examples ⚡ Random

Assuming seconds of time for "seconds" | Use [seconds of arc](#) instead

<http://www.wolframalpha.com/input/?i=2^64%2F+100x10^9+seconds> ← URL

Input interpretation:

$$\frac{2^{64}}{100 \times 10^9} \text{ seconds}$$

Unit conversions:

More

3.074×10^6 minutes

51241 hours

2135 days

305 weeks

70.19 months

Testing * operation in a CPU

Q: What if we have $(n \lg n)$ algorithm?

Assume we use a 100G-Flop CPU
can take 100B operations per sec.

- a is a 32-bit number (2³² cases)
- b is a 32-bit number (2³² cases)

When $n = 2^{32}$, with $(n \lg n)$ algorithm

$$\text{Time taken} = (2^{32} * 32 / 100 \times 10^9) \text{ sec}$$

< 2 sec!



Application in Web-Service

Suppose you code up a new web-service CoolApp

- you debugged your code, and after some time.
- you got it working, you tested it a bit
- it is quite fast

Can you release *CoolApp*?

- will it work? Or will it bomb?

If dream come true & *CoolApp*'s wildly popular?

- how fast is “quite fast”, will server die? When?

Moral of the story

Analysis of algorithms
help us make *predictions*.

Analysis of algorithms
help us *prepare for the worst case*.

Note: If operation is “quite fast”, 0.02sec / op
that’s 3min for 10,000 clicks per second.

Also, how big a load can a server take before dying?

Story of Algorithms in Action

Credit card processing centre in SG (Sci Park):

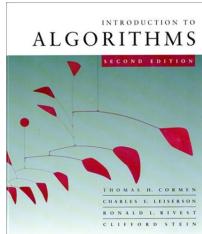
- monitor showing servers load for diff. countries,
- blue, green, yellow, orange (send SMS alert),
- RED (URGENT Alert! → deploy more servers!)



Note: Picture is NOT the real thing.

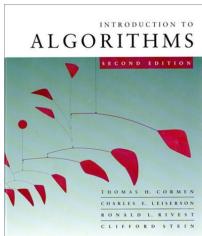
But it gives the rough idea
and “demos” my point.

Picture is NOT very good.
Will find a better one.

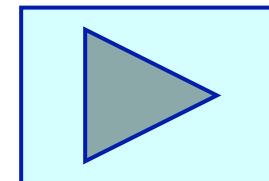
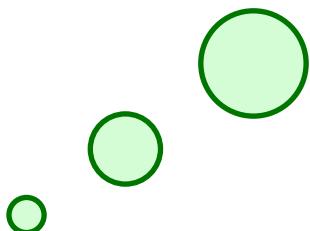


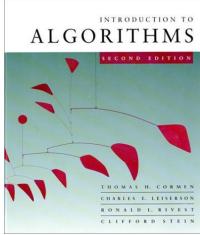
Why study algorithms and performance?

- Algorithms help us to understand *scalability*.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a *language* for talking about program behavior.
- Performance is the *currency* of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

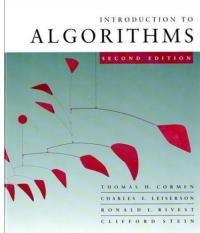


*Introduction to some
fun AA patterns*





*Now analyze some
algorithms and fit
into AA patterns*



The problem of sorting

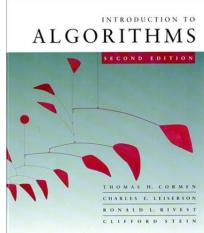
Input: sequence $\langle a_1, a_2, \dots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \dots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

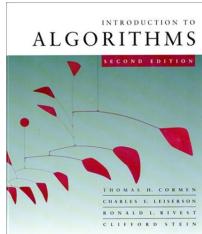
Output: 2 3 4 6 8 9



Insertion sort

“pseudocode”

```
INSERTION-SORT( $A, n$ )      ▷  $A[1 \dots n]$ 
  for  $j \leftarrow 2$  to  $n$ 
    do  $key \leftarrow A[j]$ 
         $i \leftarrow j - 1$ 
        while  $i > 0$  and  $A[i] > key$ 
          do  $A[i+1] \leftarrow A[i]$ 
               $i \leftarrow i - 1$ 
     $A[i+1] = key$ 
```

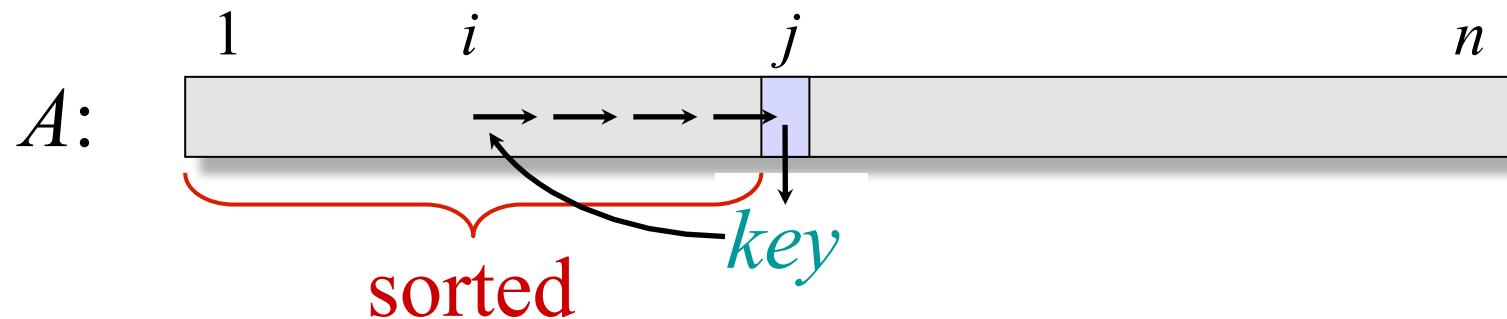


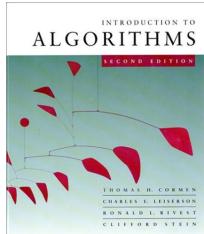
Insertion sort

“pseudocode”

{

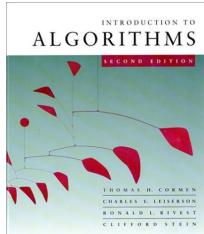
```
INSERTION-SORT( $A, n$ )      ▷  $A[1 \dots n]$ 
    for  $j \leftarrow 2$  to  $n$ 
        do  $key \leftarrow A[j]$ 
             $i \leftarrow j - 1$ 
            while  $i > 0$  and  $A[i] > key$ 
                do  $A[i+1] \leftarrow A[i]$ 
                     $i \leftarrow i - 1$ 
             $A[i+1] = key$ 
```





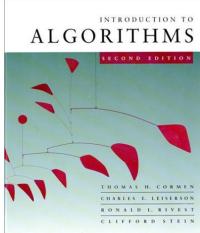
Example of insertion sort

8 2 4 9 3 6

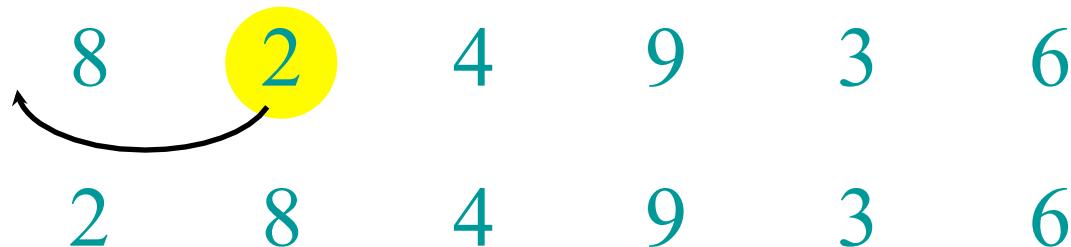


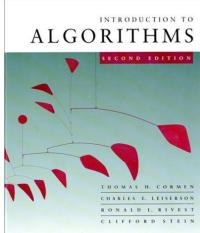
Example of insertion sort





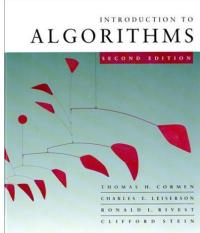
Example of insertion sort



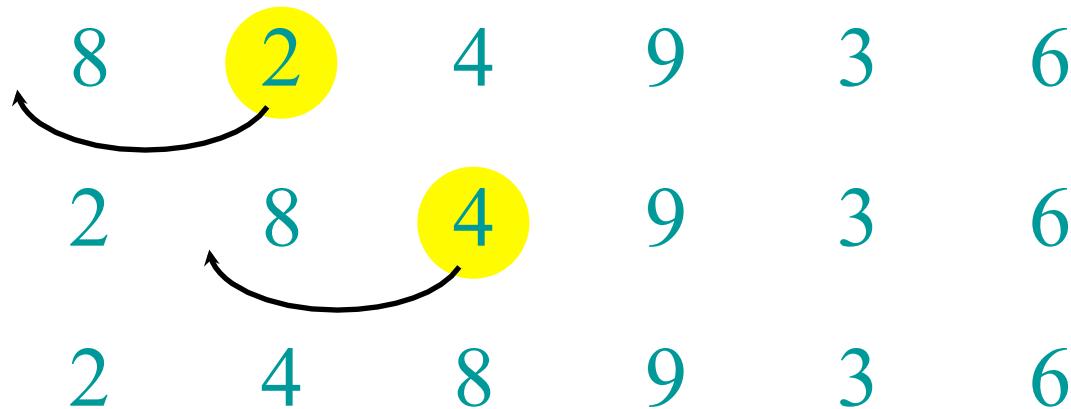


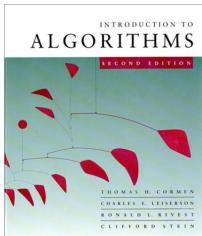
Example of insertion sort



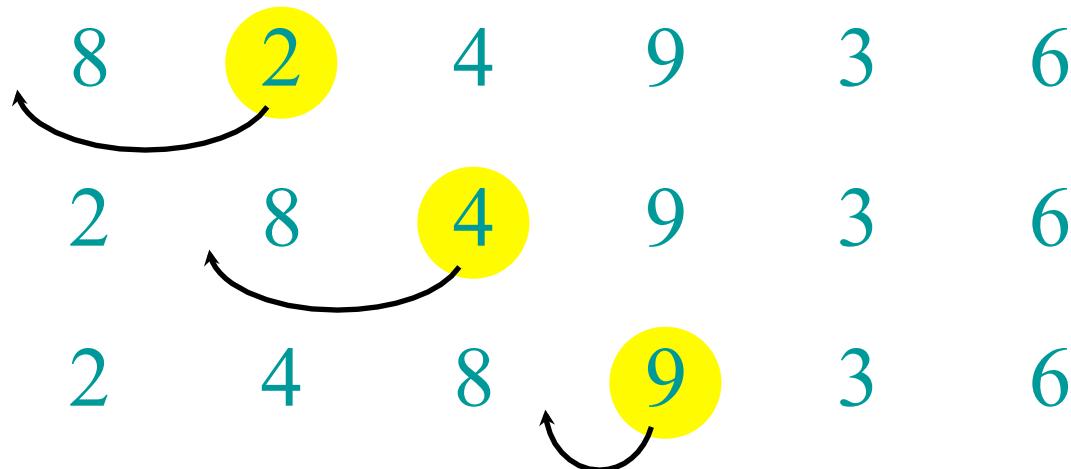


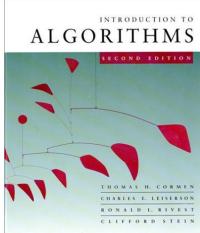
Example of insertion sort



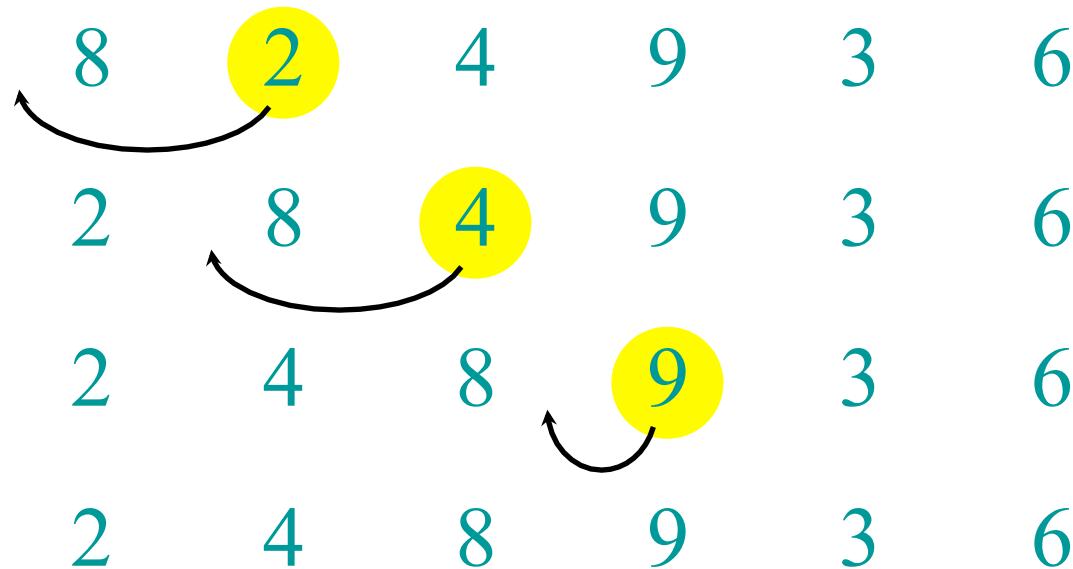


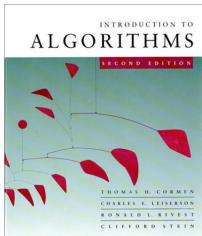
Example of insertion sort



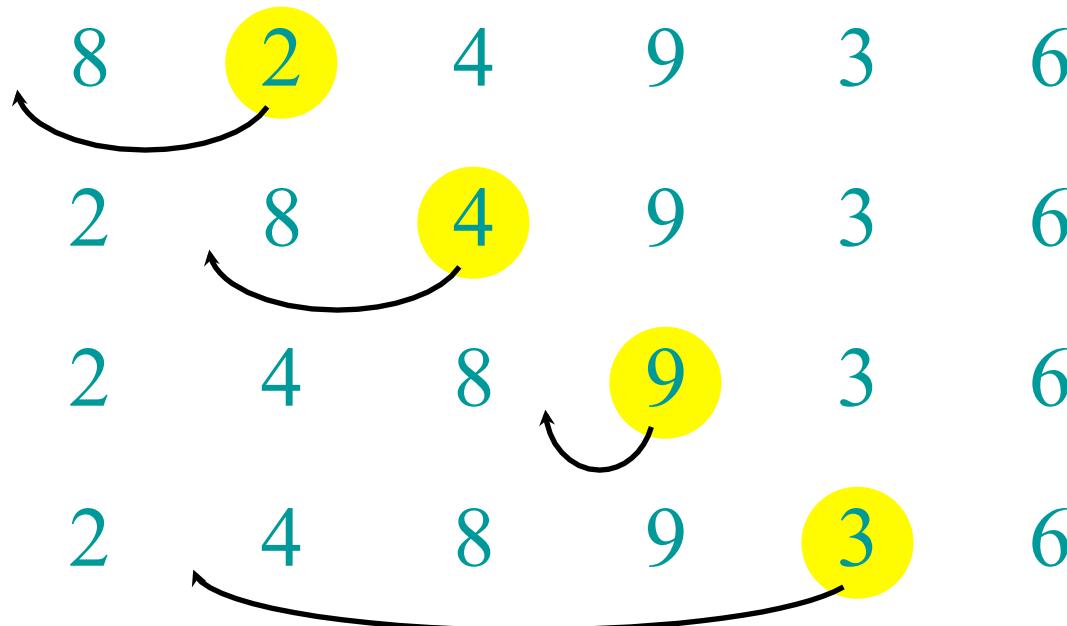


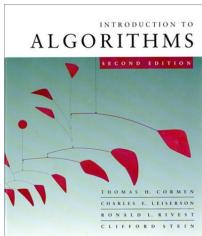
Example of insertion sort



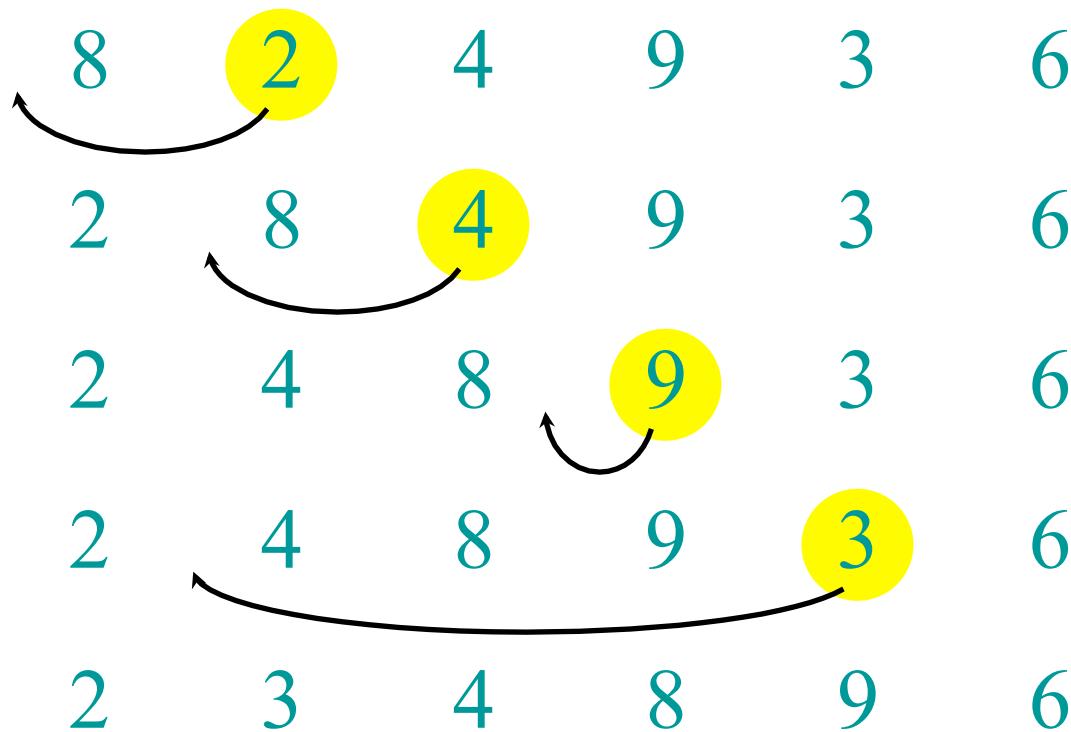


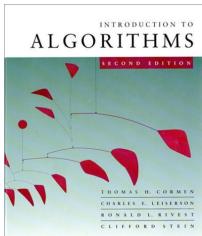
Example of insertion sort



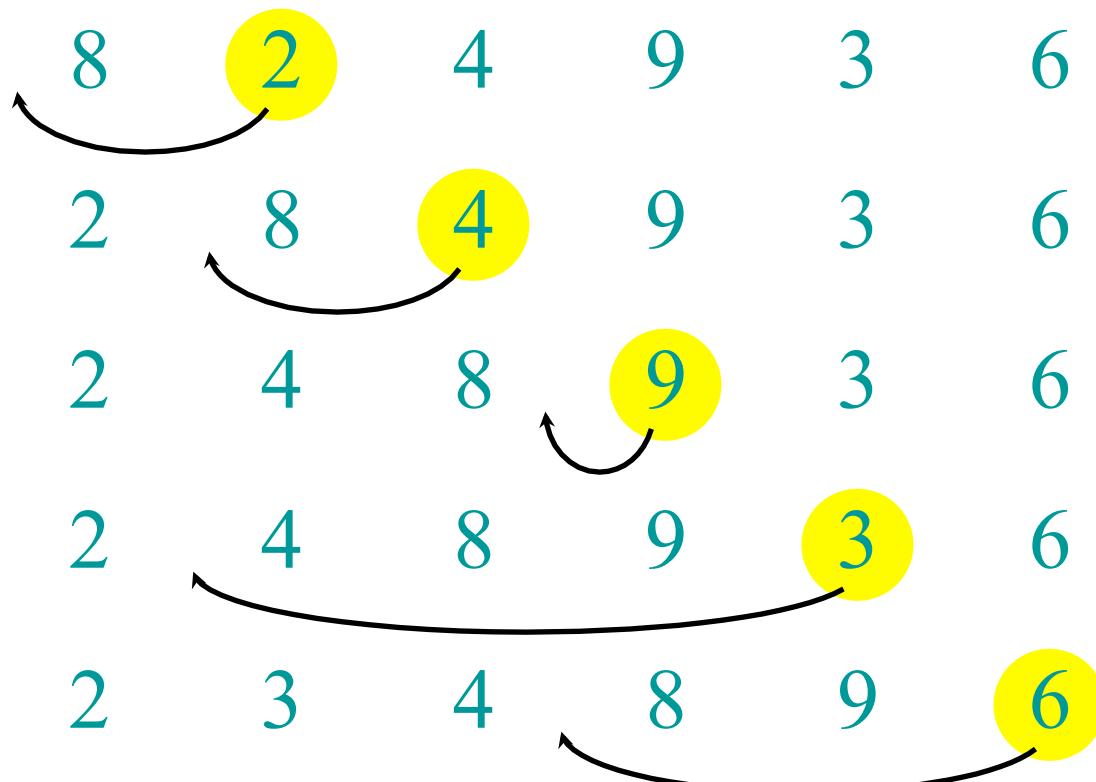


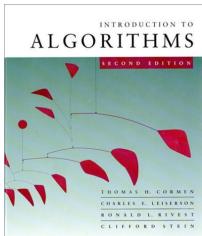
Example of insertion sort



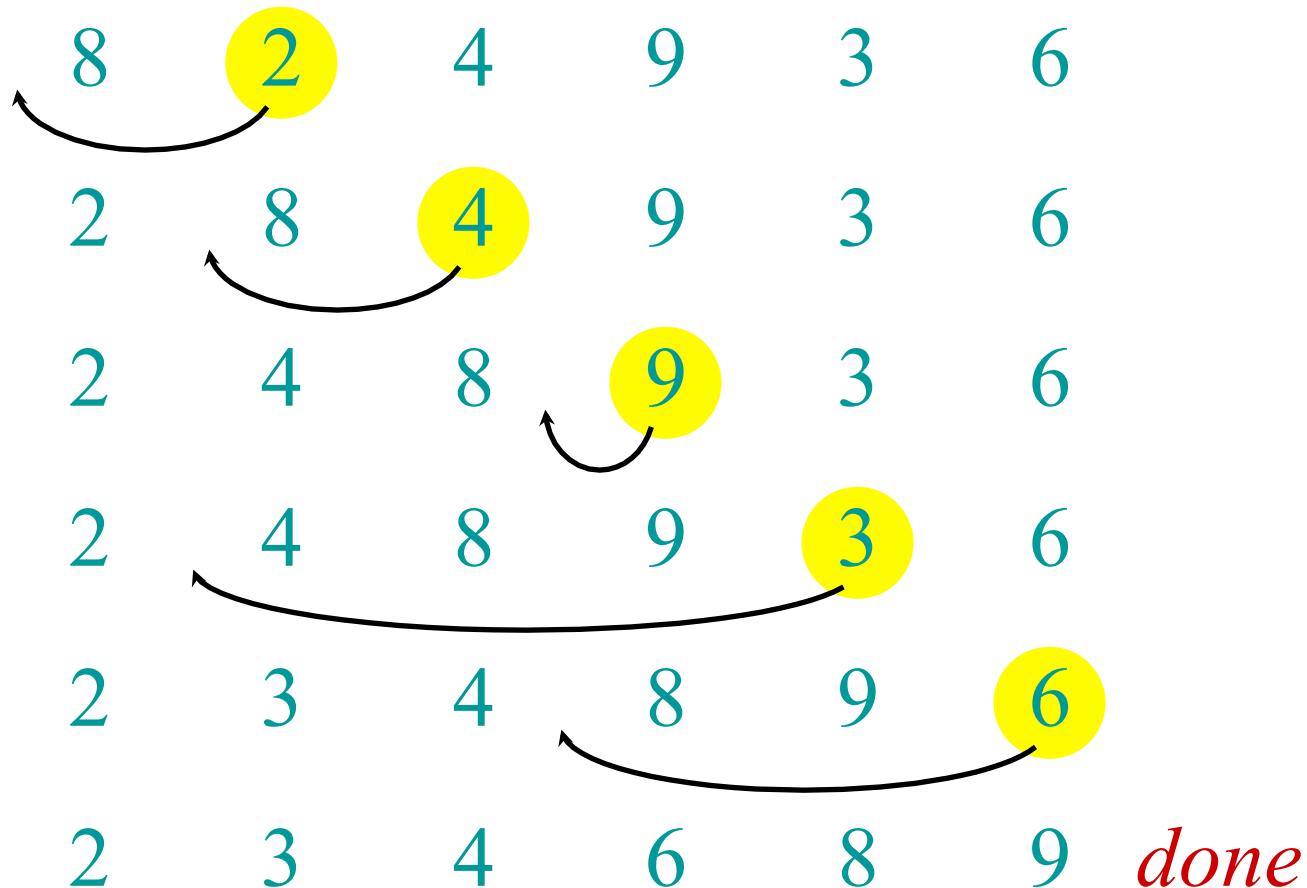


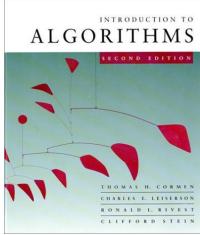
Example of insertion sort





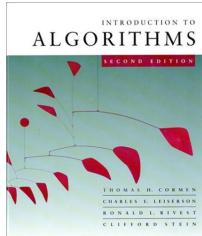
Example of insertion sort





Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.



Kinds of analyses

Worst-case: (usually)

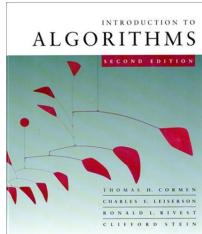
- $T(n)$ = maximum time of algorithm on any input of size n .

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm over all inputs of size n .
- Need assumption of statistical distribution of inputs.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on *some* input.



Machine-independent time

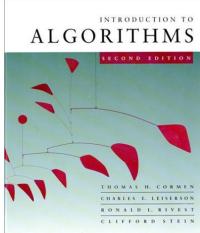
What is insertion sort's worst-case time?

- It depends on the speed of our computer:
 - relative speed (on the same machine),
 - absolute speed (on different machines).

BIG IDEA:

- Ignore machine-dependent constants.
- Look at *growth* of $T(n)$ as $n \rightarrow \infty$.

“Asymptotic Analysis”



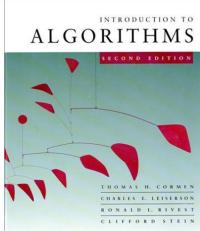
Θ -notation

Math:

$\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$

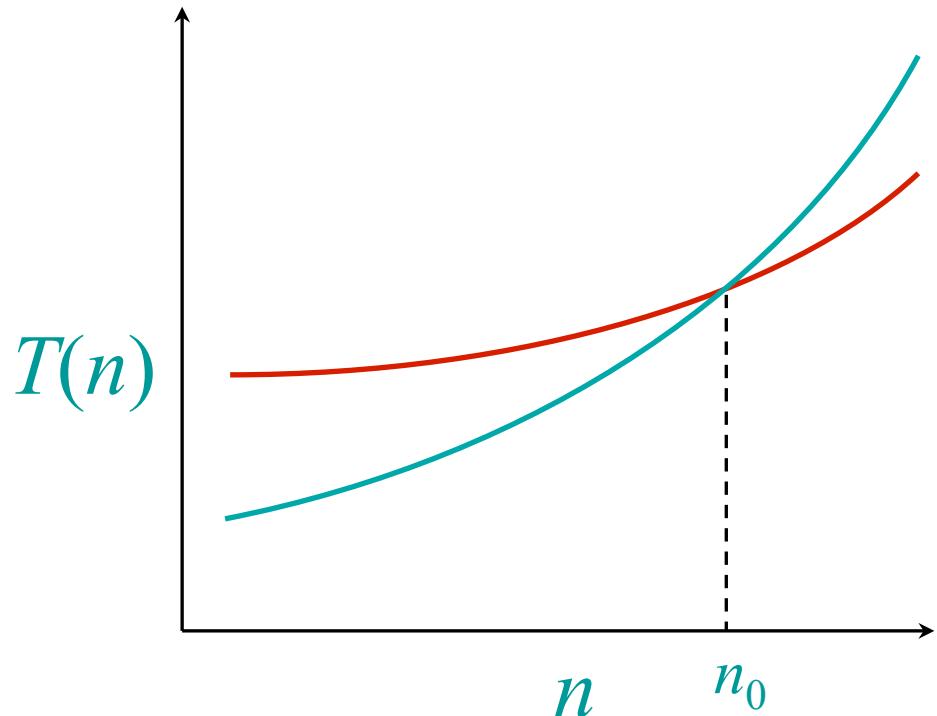
Engineering:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

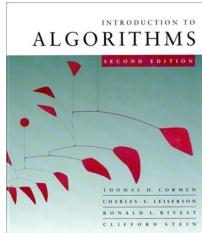


Asymptotic performance

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.

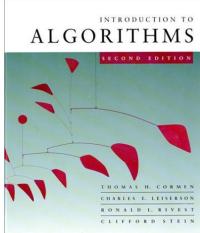


Comparing $100n^2$ vs $0.01n^3$

n	$100n^2$	$0.01n^3$	$Diff$	$Ratio$
1	100	0.01	-99.99	10,000
10	10,000	10	-9,990	1,000
100	1,000,000	10,000	-990,000	100
1,000	100,000,000	10,000,000	-90,000,000	10
10,000	10,000,000,000	10,000,000,000	0	1
10,001	10,002,000,100	10,003,000,300	1,000,200.01	0.9999
50,000	2.5E+11	1.25E+12	1E+12	0.2000

Higher order terms dominates
(ignore lower order terms)

Can ignore
constant factors



Insertion sort analysis

Worst case: Input reverse sorted.

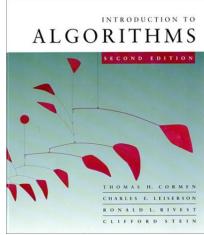
$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2) \quad [\text{arithmetic series}]$$

Average case: All permutations equally likely.

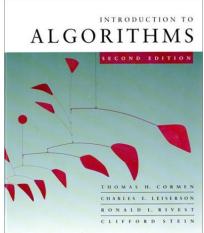
$$T(n) = \sum_{j=2}^n \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n .
- Not at all, for large n .



What's the AA pattern for Insertion Sort (worst-case)?



More sorting algorithm: Selection Sort

Sorting: Problem and Algorithms

❖ Problem: Sorting

- ❑ Take a list of n numbers and rearrange them in increasing order

❖ Algorithms:

- ❑ Selection Sort
- ❑ Insertion Sort
- ❑ Bubble Sort
- ❑ Merge Sort
- ❑ Quicksort

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n \lg n)$

$\Theta(n \lg n)^{**}$

Some slides from
UIT2201

Not covered
in the course
UIT2201

*** average case*

Selection sort

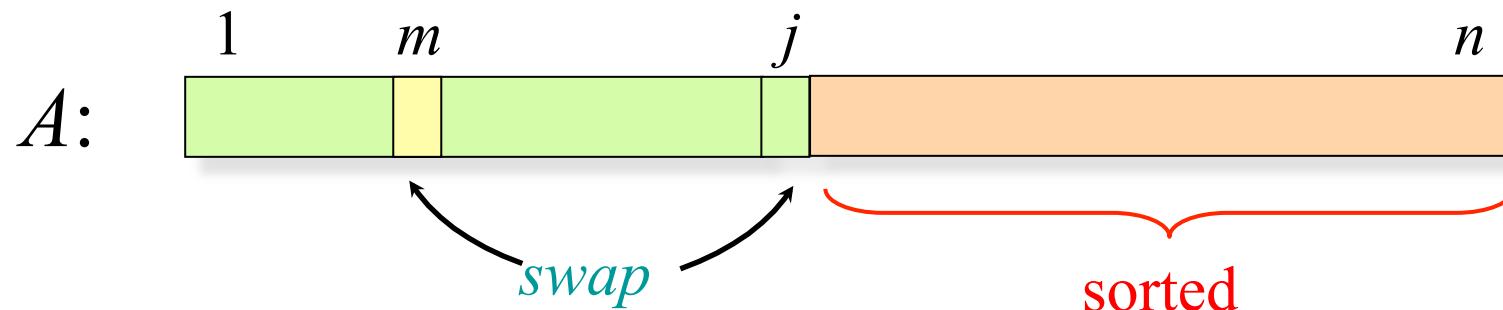
❖ Idea behind the algorithm...

❑ Repeatedly

- ◆ *find the largest number in unsorted section*
- ◆ *Swap it to the end (the sorted section)*

❑ Re-uses the Find-Max algorithm

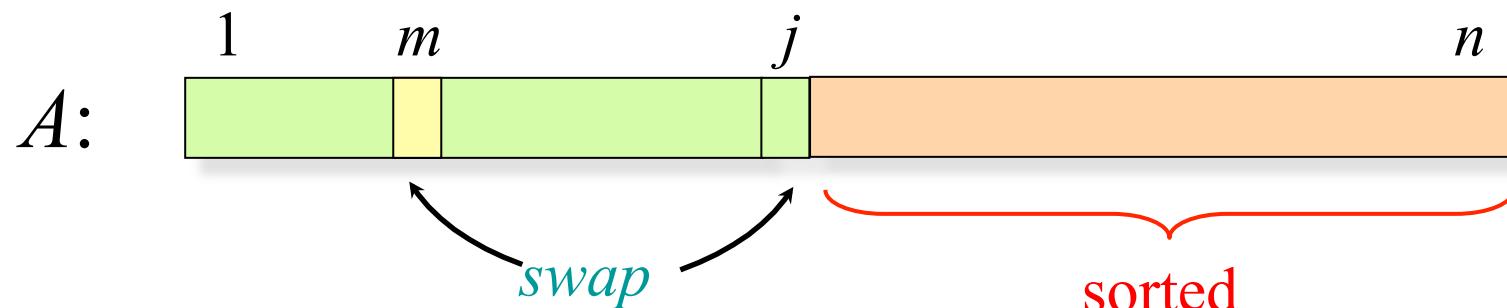
$A[m]$ is largest among $A[1..j]$



Selection Sort Algorithm (pseudo-code)

```
Selection-Sort(A, n) ;  
begin  
    j ← n;  
    while (j > 1) do  
        m ← Find-Max(A, j) ;  
        swap(A[m] ,A[j]) ;  
        j ← j - 1;  
    endwhile  
end;
```

$A[m]$ is largest among $A[1..j]$



Example of selection sort

6	10	13	5	8
---	----	----	---	---

m j
 \longleftrightarrow
swap

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

$j \leftarrow$

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

m j \leftarrow
 \longleftrightarrow
swap

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

6	5	8	10	13
---	---	---	----	----

$j \leftarrow$

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

6	5	8	10	13
---	---	---	----	----

$j \leftarrow$
 m

swap

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

6	5	8	10	13
---	---	---	----	----

6	5	8	10	13
---	---	---	----	----

j ← →

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

6	5	8	10	13
---	---	---	----	----

6	5	8	10	13
---	---	---	----	----

m j \leftarrow ●
↔
swap

Example of selection sort

6	10	13	5	8
---	----	----	---	---

6	10	8	5	13
---	----	---	---	----

6	5	8	10	13
---	---	---	----	----

6	5	8	10	13
---	---	---	----	----

5	6	8	10	13
---	---	---	----	----

Done. $j \leftarrow$

What about the time complexity?

❖ Dominant operation: *comparisons*



4 comparisons



3 comparisons



2 comparisons



1 comparisons



Done. $j \leftarrow$

Analysis of Selection Sort

Find-Max for j numbers takes $(j-1)$ comparisons

❖ When sorting n numbers,

- ❑ $(n-1)$ comparisons in iteration 1 (when $j = n$)
- ❑ $(n-2)$ comparisons in iteration 2 (when $j = n-1$)
- ❑ $(n-3)$ comparisons in iteration 3 (when $j = n-2$)
- ❑
- ❑ 2 comparisons in iteration $(n-2)$ (when $j = 3$)
- ❑ 1 comparisons in iteration $(n-1)$ (when $j = 2$)

❖ Total number of comparisons:

$$\begin{aligned} \text{❑ Cost} &= (n-1) + (n-2) + \dots + 2 + 1 = n(n-1)/2 \\ &= \Theta(n^2) \end{aligned}$$

Analysis of Selection Sort: Summary

- ❖ **Time complexity:** $\Theta(n^2)$
 - Comparisons: $n(n-1)/2$
 - Exchanges: n (swapping largest into place)
 - Overall time complexity: $\Theta(n^2)$
- ❖ **Space complexity:** $\Theta(n)$
 - $\Theta(n)$ – space for input sequence,
plus a few variables.

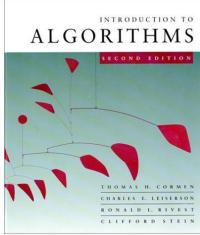
Selection Sort:

Time complexity: $T(n) = \Theta(n^2)$

Space complexity: $S(n) = \Theta(n)$

AA Pattern

What is the AA Pattern for Selection Sort?



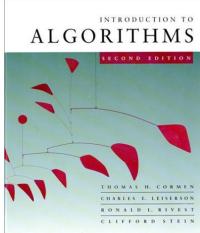
Merge sort

MERGE-SORT $A[1 \dots n]$

1. If $n = 1$, done.
2. Recursively sort $A[1 \dots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1 \dots n]$.
3. “*Merge*” the 2 sorted lists.

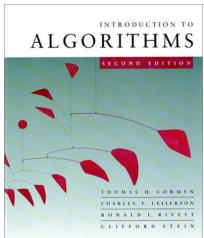
Key subroutine: MERGE

Mergesort is $\Theta(n \lg n)$ (*shown later*).
[Also covered in CS2010/CS2020)]

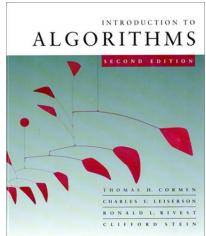


Observations about Sorting

- $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.
- Go test it out for yourself!



Graph Pattern?



Unweighted graphs

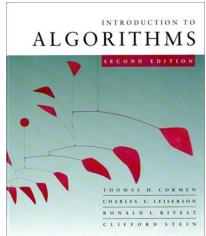
Suppose that $w(u, v) = 1$ for all $(u, v) \in E$.
Can use BFS to find shortest paths.

- Use a simple FIFO queue.

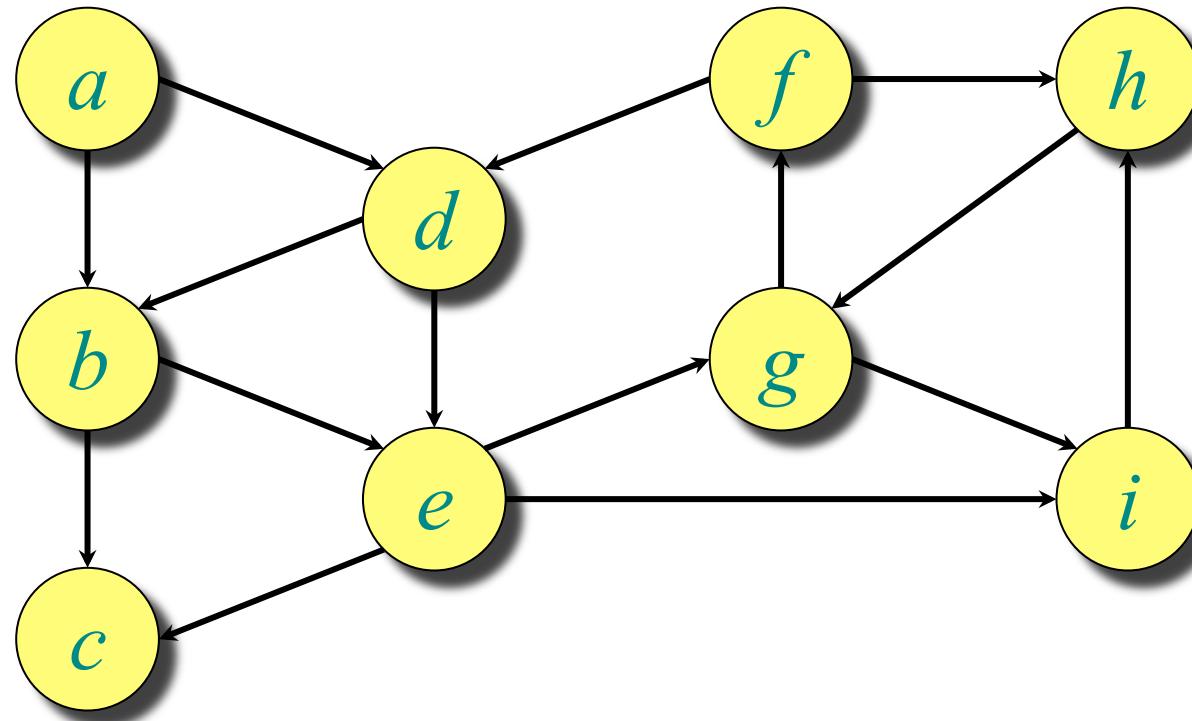
Breadth-first search

```
while  $Q \neq \emptyset$ 
    do  $u \leftarrow \text{DEQUEUE}(Q)$ 
        for each  $v \in \text{Adj}[u]$ 
            do if  $d[v] = \infty$ 
                then  $d[v] \leftarrow d[u] + 1$ 
                    ENQUEUE( $Q, v$ )
```

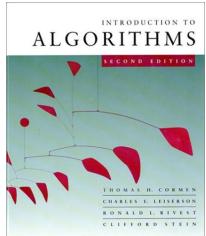
Analysis: Time = $O(V + E)$.



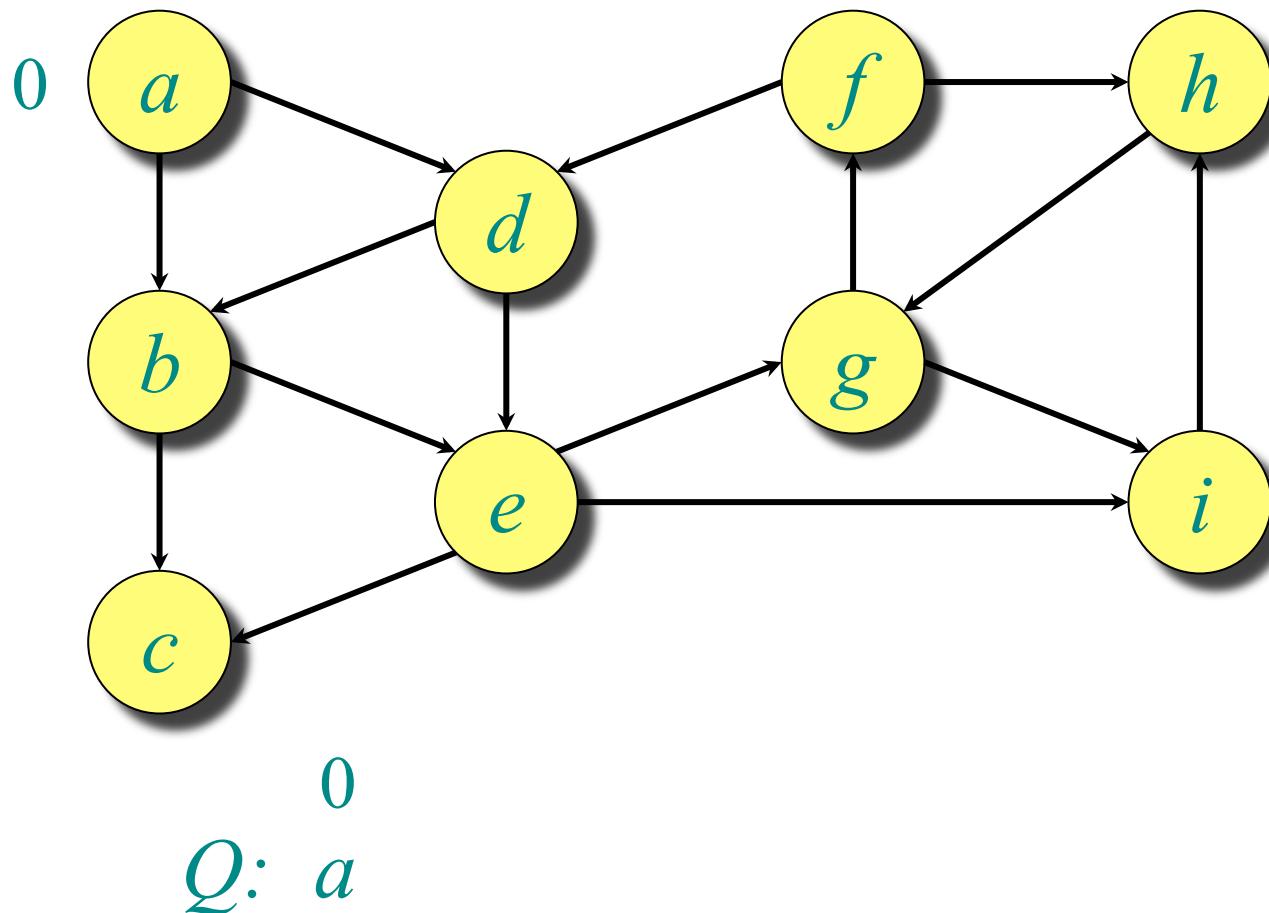
Example of breadth-first search

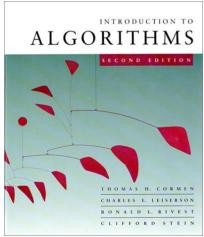


Q:

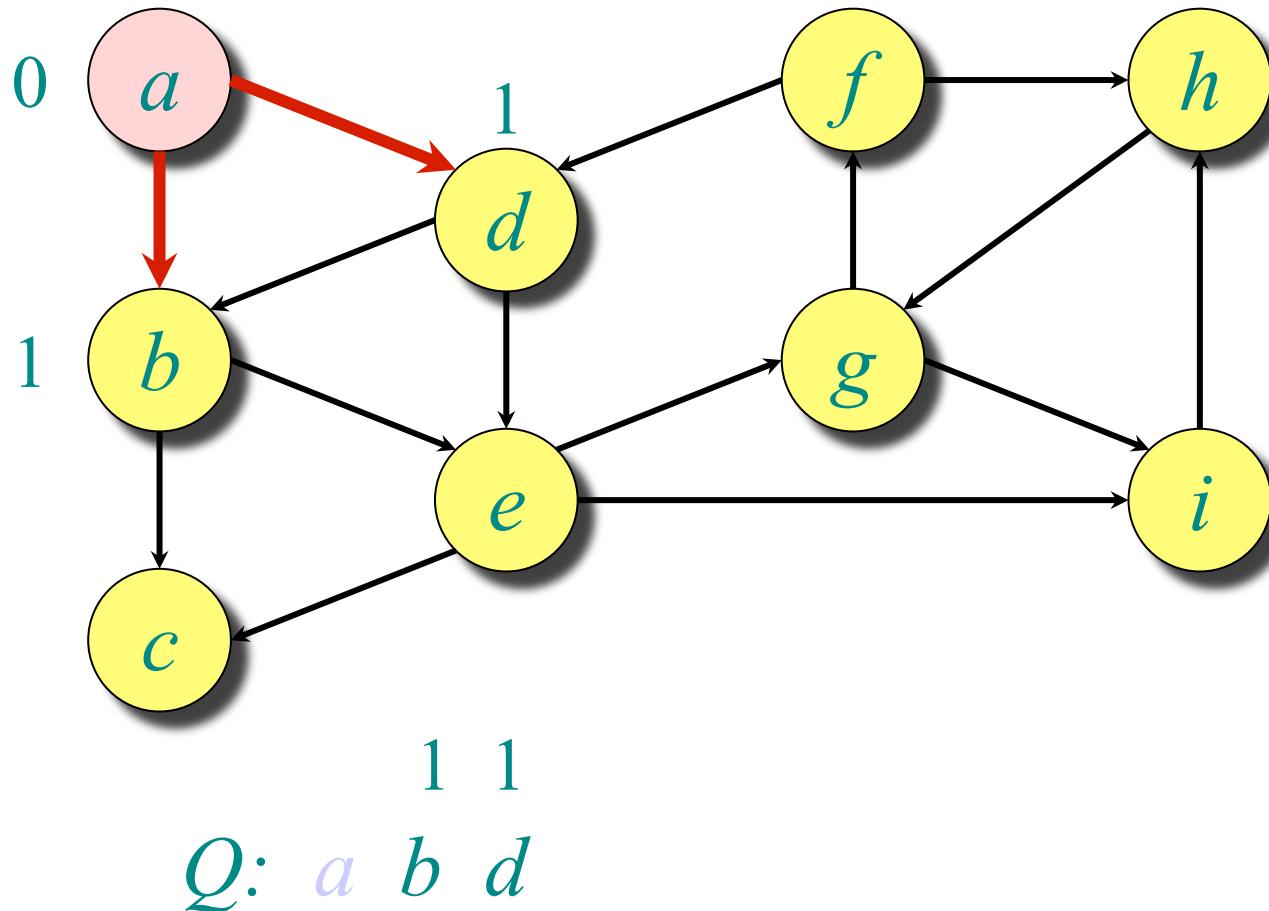


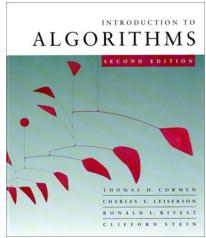
Example of breadth-first search



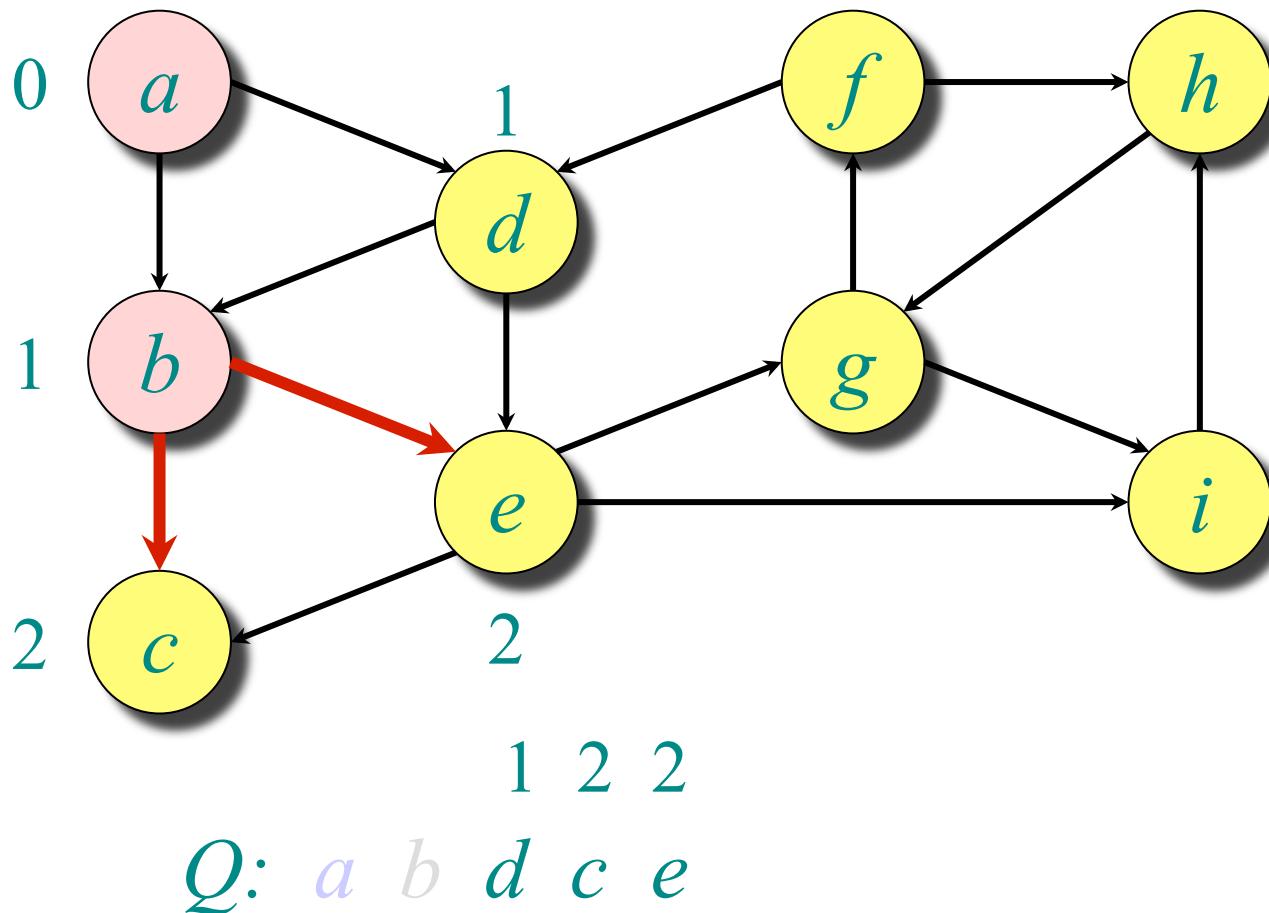


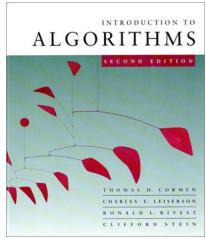
Example of breadth-first search



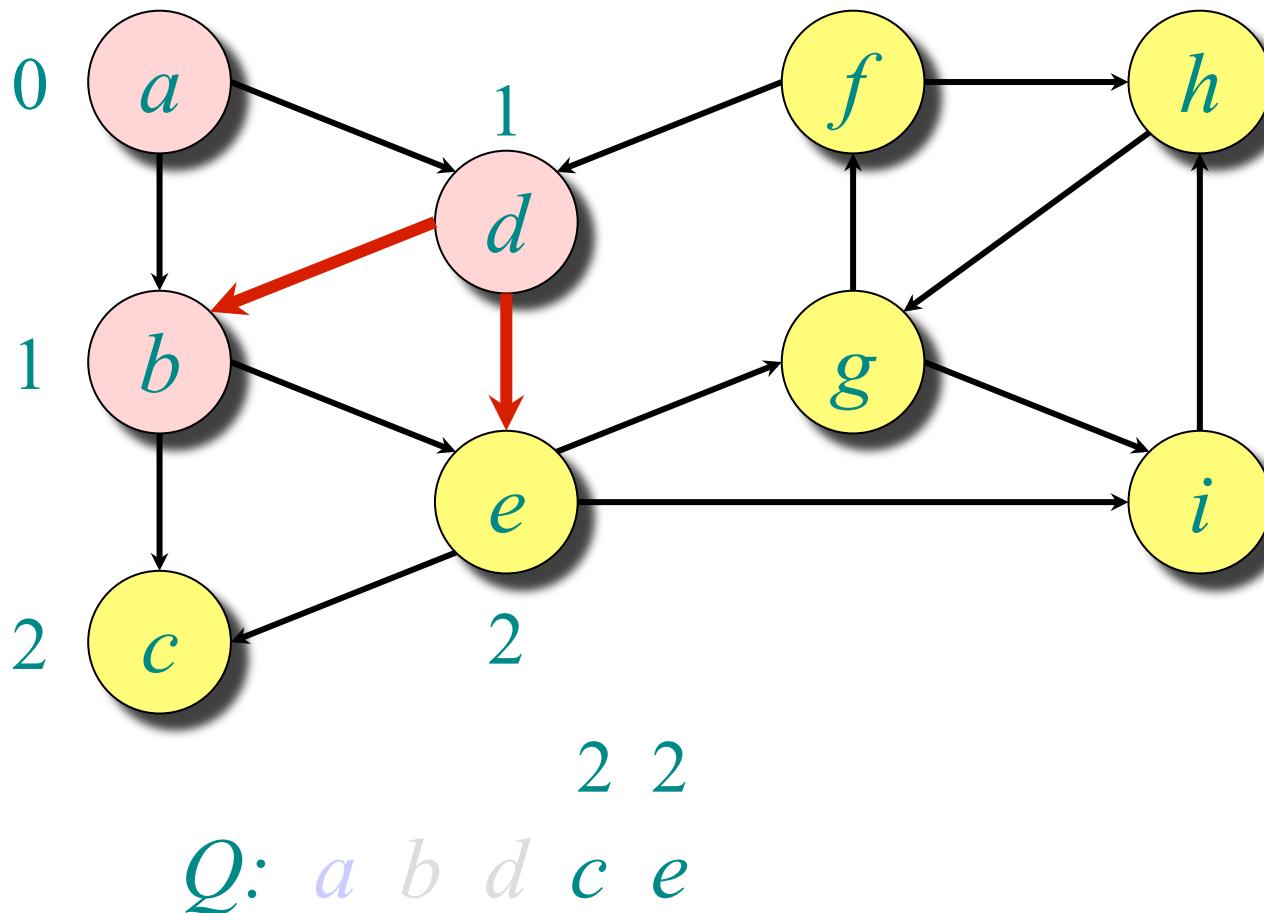


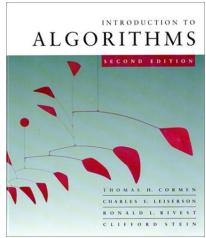
Example of breadth-first search



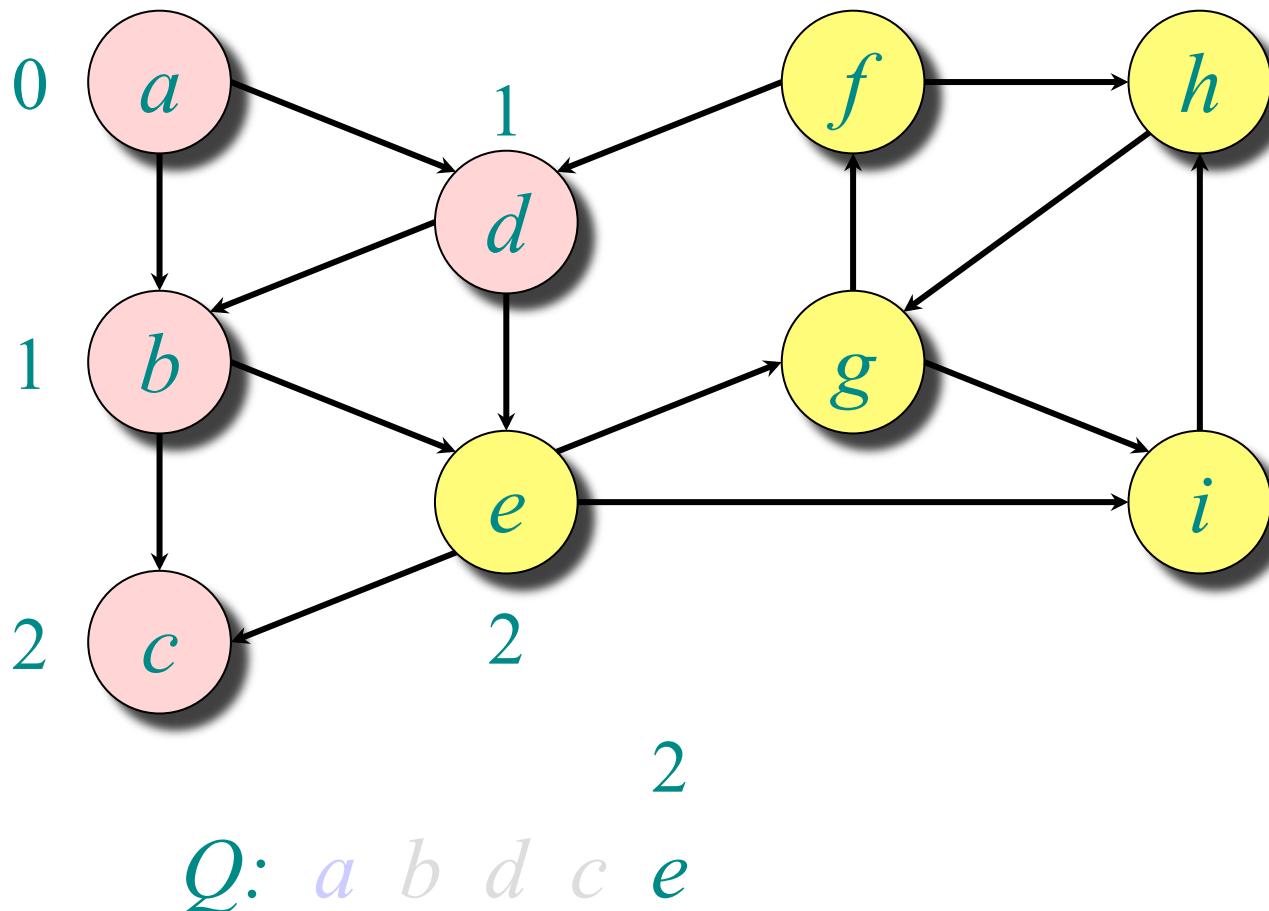


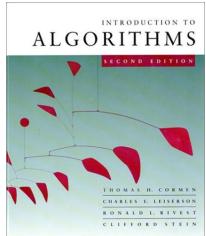
Example of breadth-first search



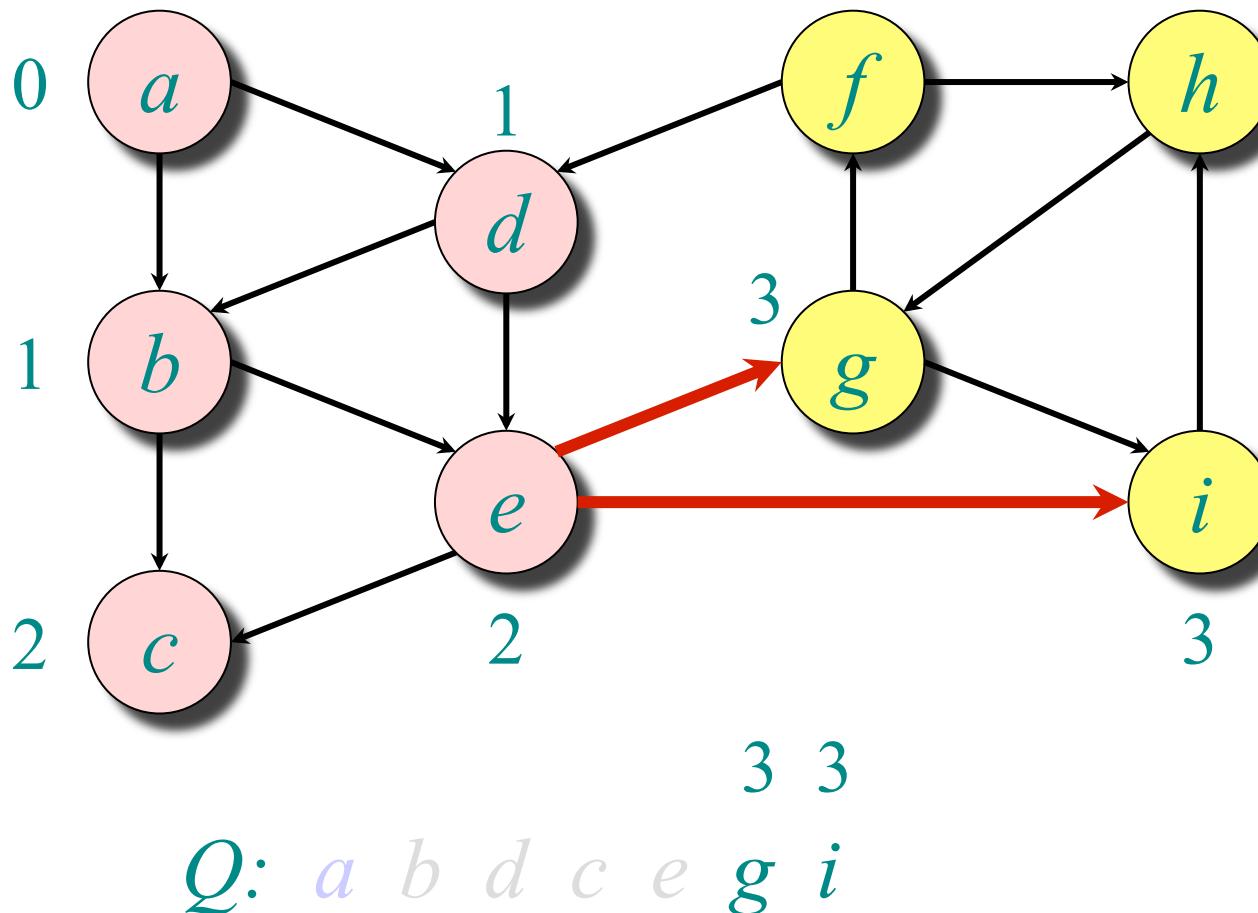


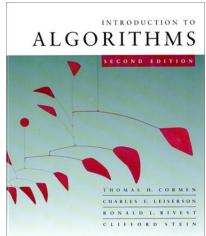
Example of breadth-first search



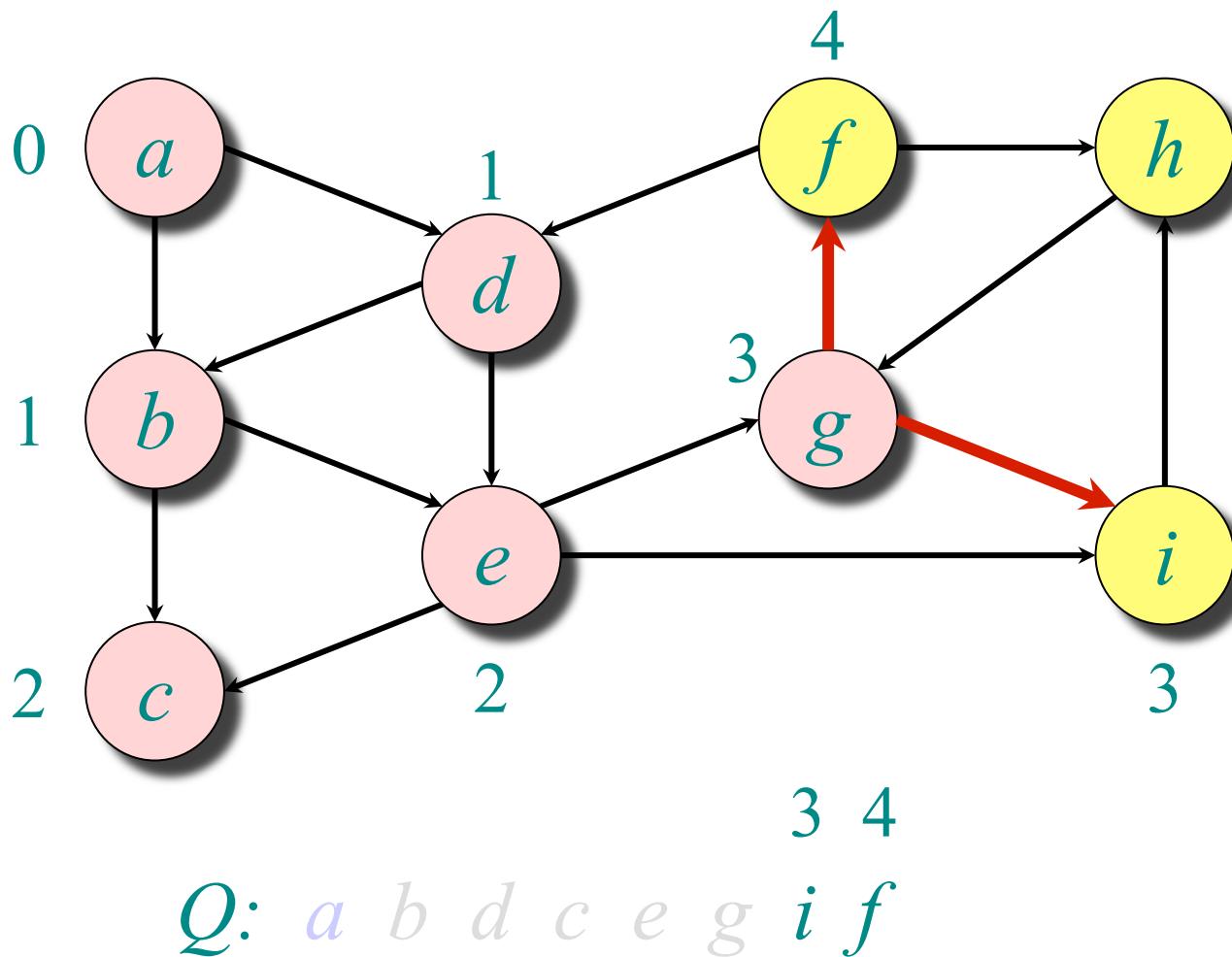


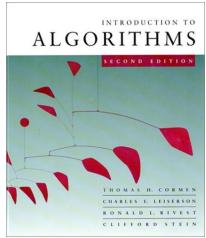
Example of breadth-first search



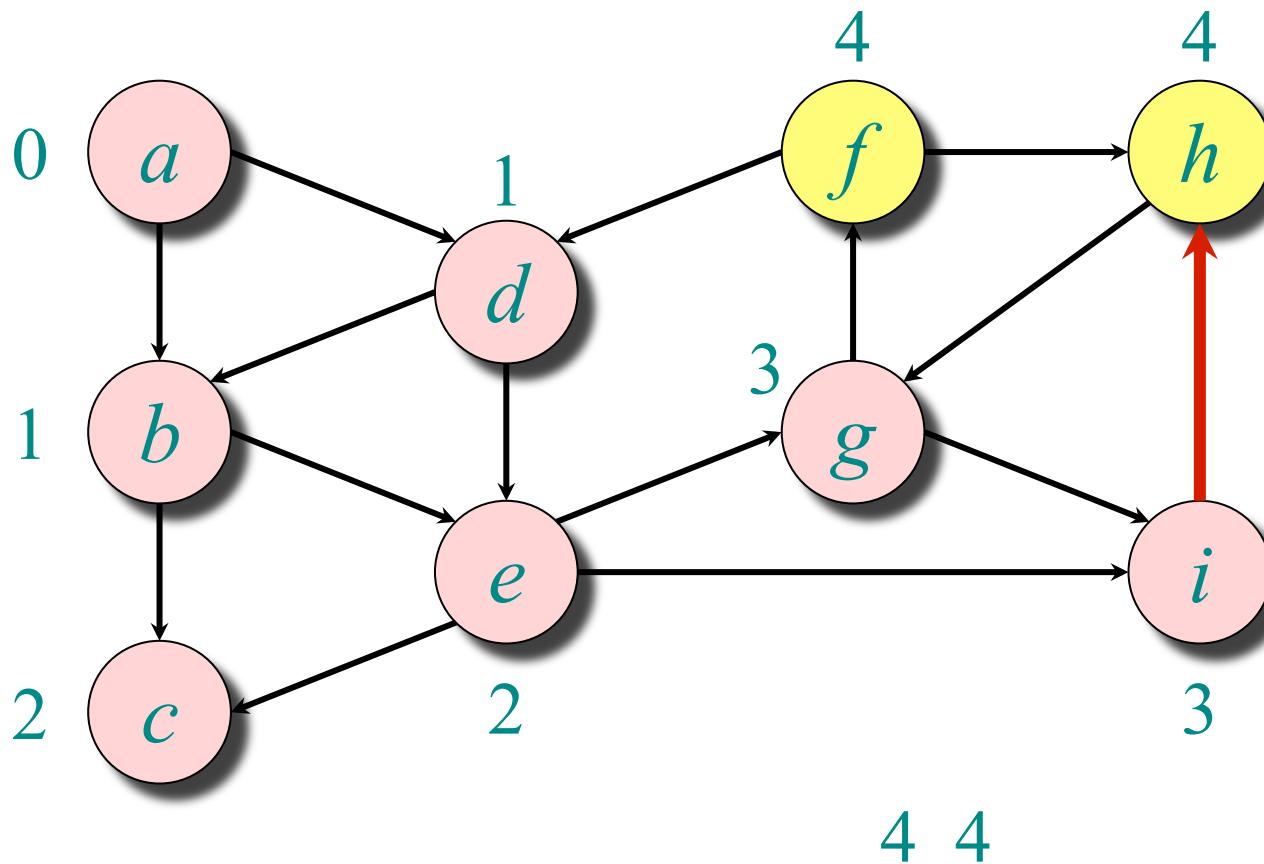


Example of breadth-first search

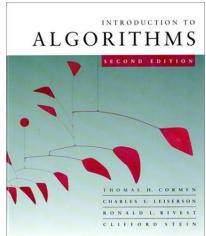




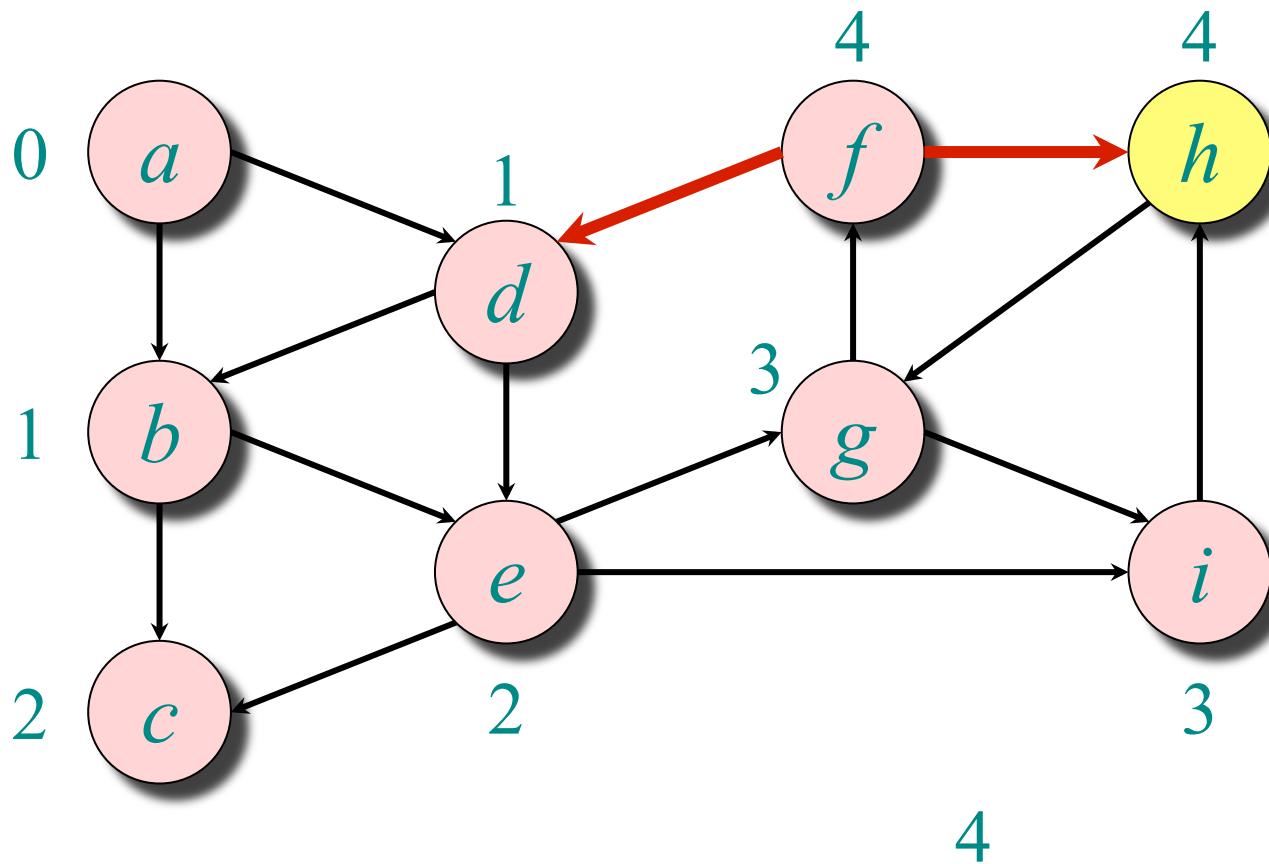
Example of breadth-first search



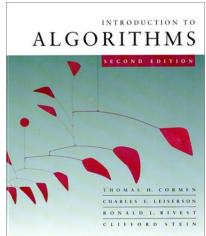
$Q: a \ b \ d \ c \ e \ g \ i \ f \ h$



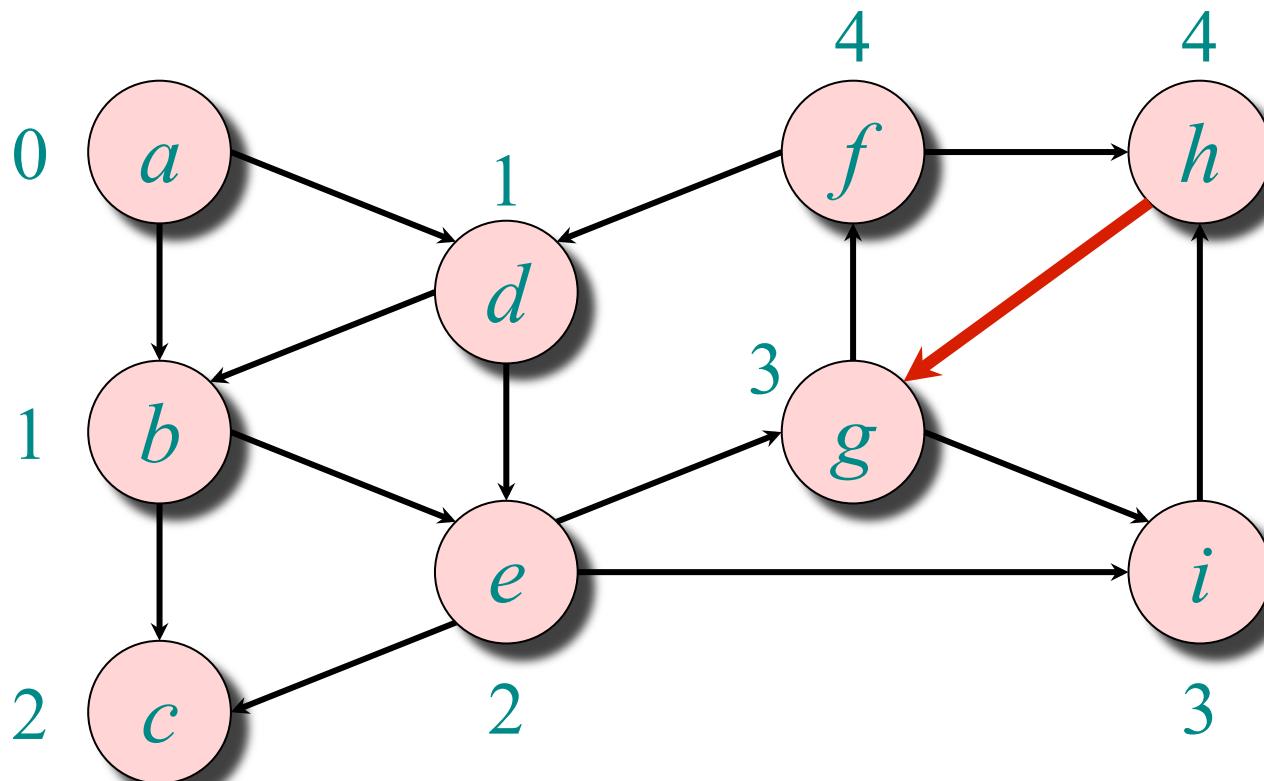
Example of breadth-first search



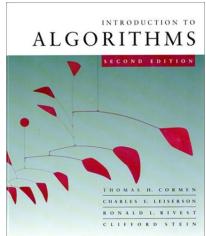
$Q: a \ b \ d \ c \ e \ g \ i \ f \ h$



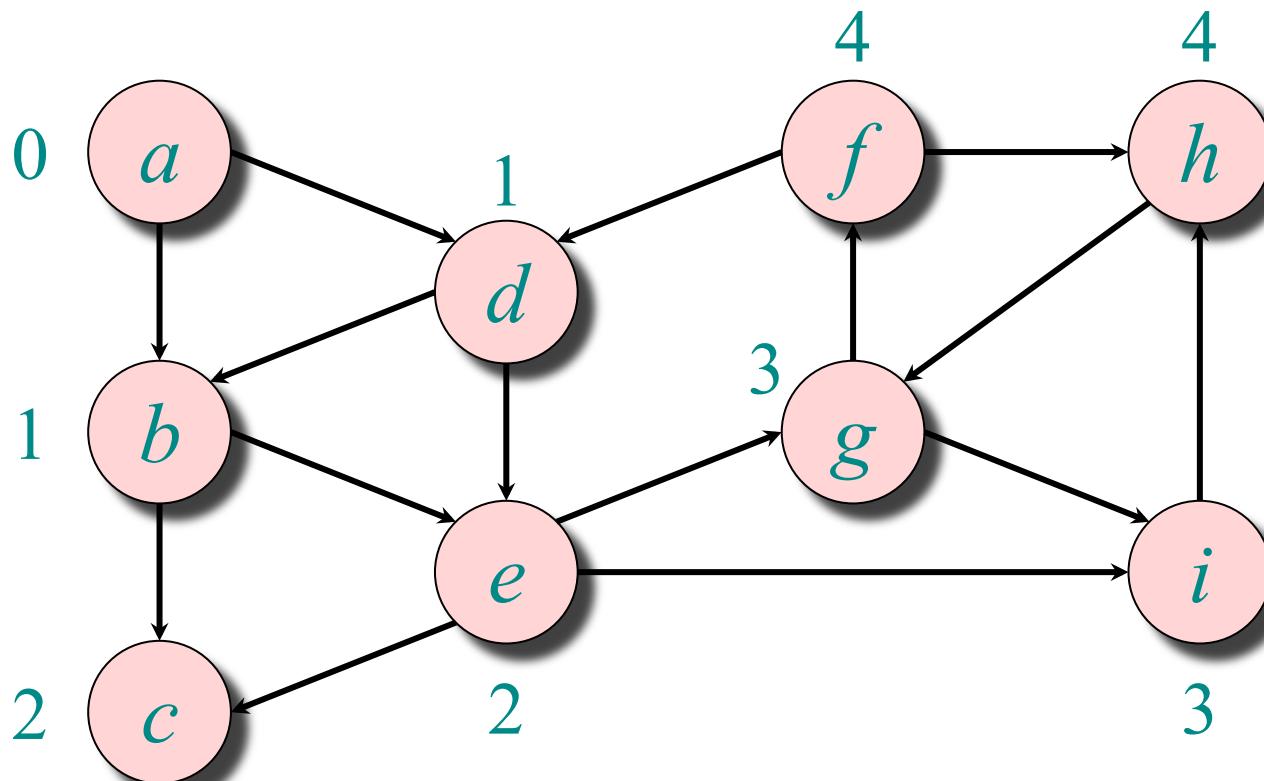
Example of breadth-first search



Q: *a b d c e g i f h*



Example of breadth-first search



Thank you.

Q & A



School *of* Computing

CS3230 Algorithm Analysis

“Math for AA”

❑ Lecture Topics and Readings

- | | |
|-----------------------------------|----------------|
| ❖ Simple Algorithm Analysis | CS1020/2010 |
| ❖ Algorithm Analysis Pattern | [LHW] A&E |
| ❖ Asymptotic Notations | [CLRS]-C3 |
| ❖ Functions & Summations (Review) | [CLRS]-C3, App |



School *of* Computing

Quickly get up to speed.



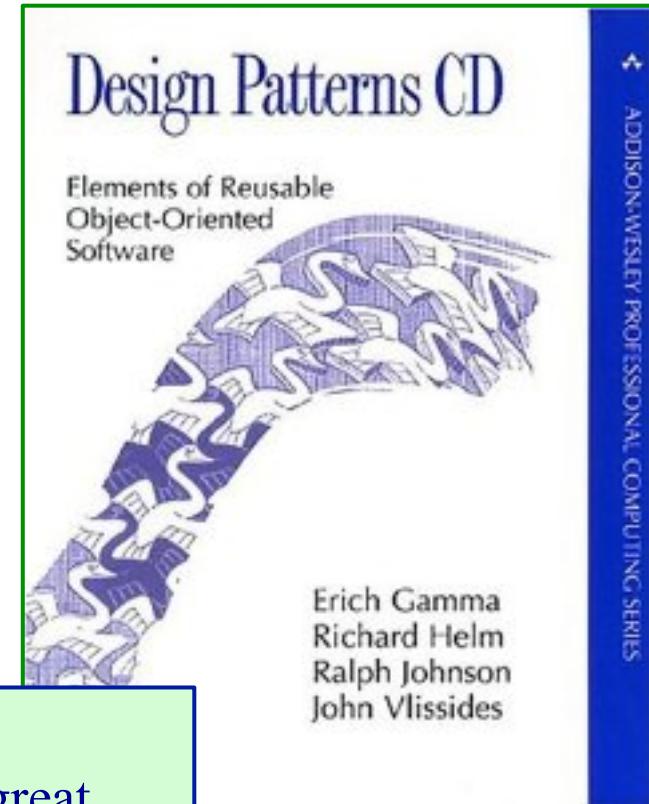
Software Design Patterns

- Started with GoF book, 1995
 - ❖ Gave list of 23 patterns

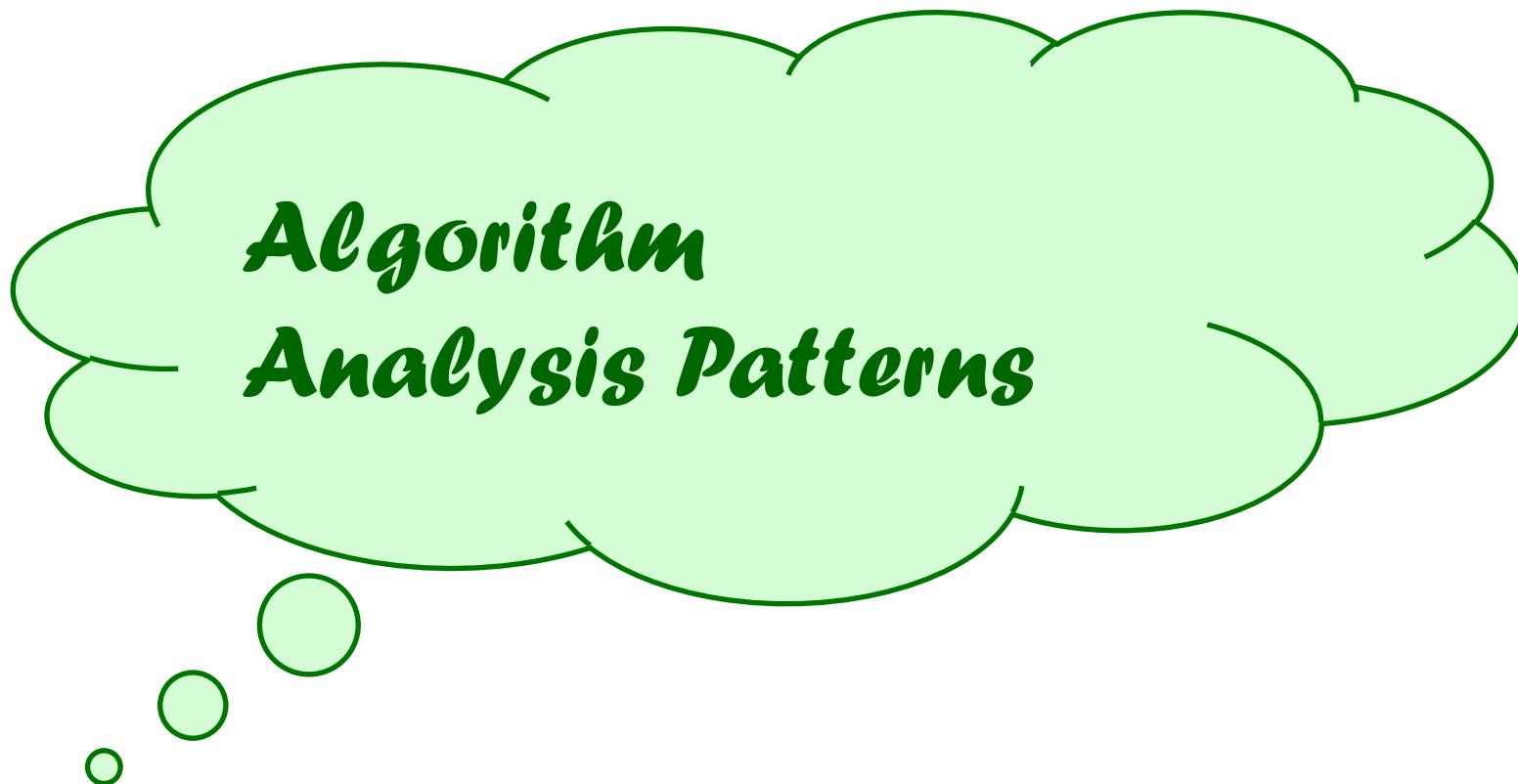
Design Patterns CD: Elements of Reusable Object-Oriented Software (Professional Computing) [CD-ROM]

[Erich Gamma](#)  (Author), [Richard Helm](#)  (Author), [Ralph Johnson](#)  (Author), [John Vlissides](#)  (Author)

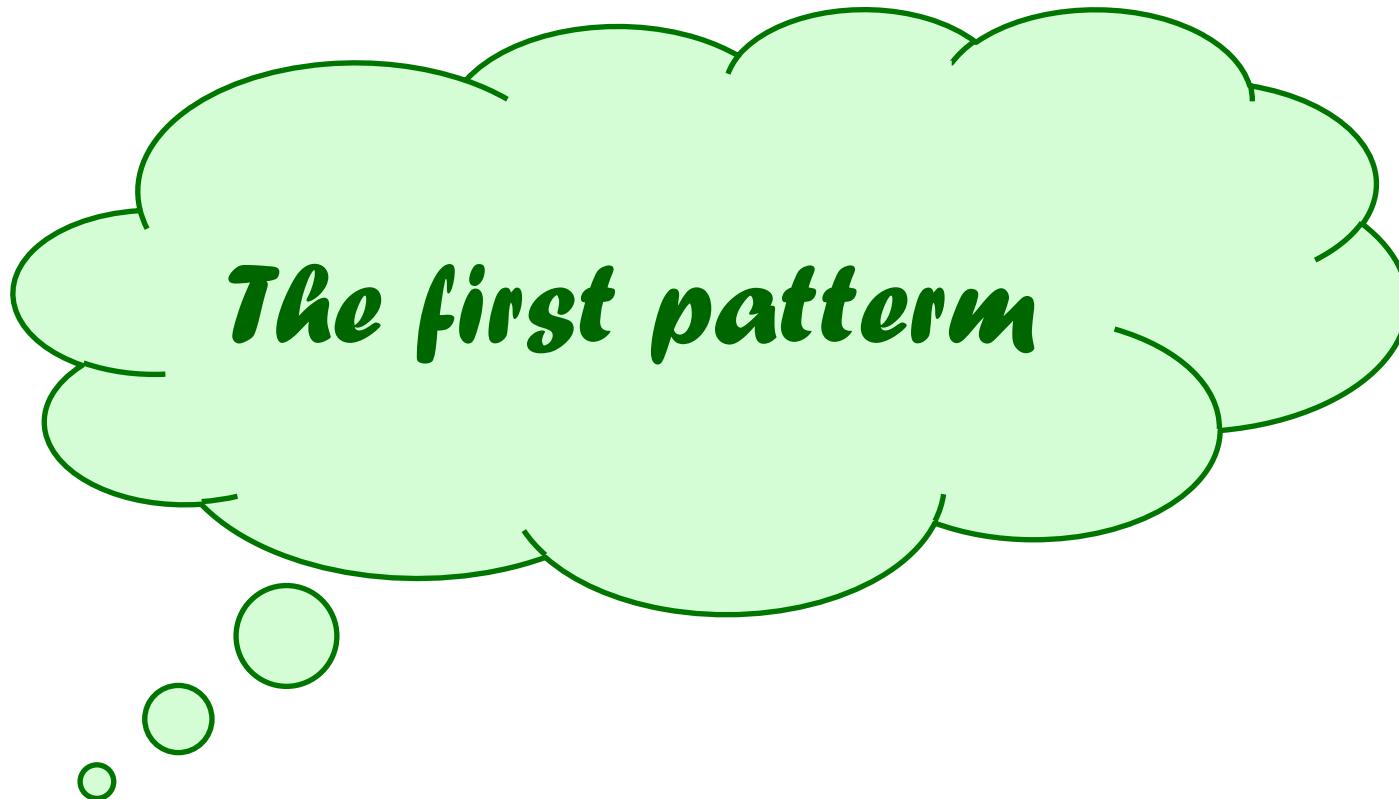
“Published in 1995, Design Patterns: Elements of Reusable Object-Oriented Software has elicited a great deal of praise from the press and readers. The 23 patterns contained in the book have become an essential resource for anyone developing reusable software designs.”



How about...



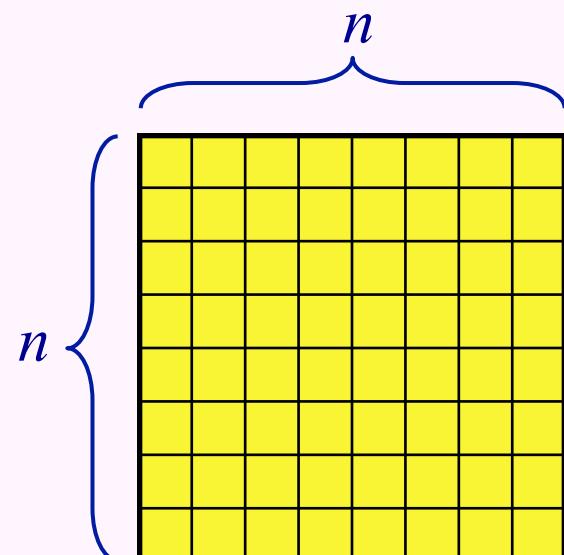
Algorithm-Analysis Pattern?



Algorithm Analysis Pattern

AA-Pattern: 2D-Box

$$T(n) = \sum_{k=1}^n cn = \sum_{k=1}^n \left(\sum_{j=1}^n c \right) = \Theta(n^2)$$



Code Pattern

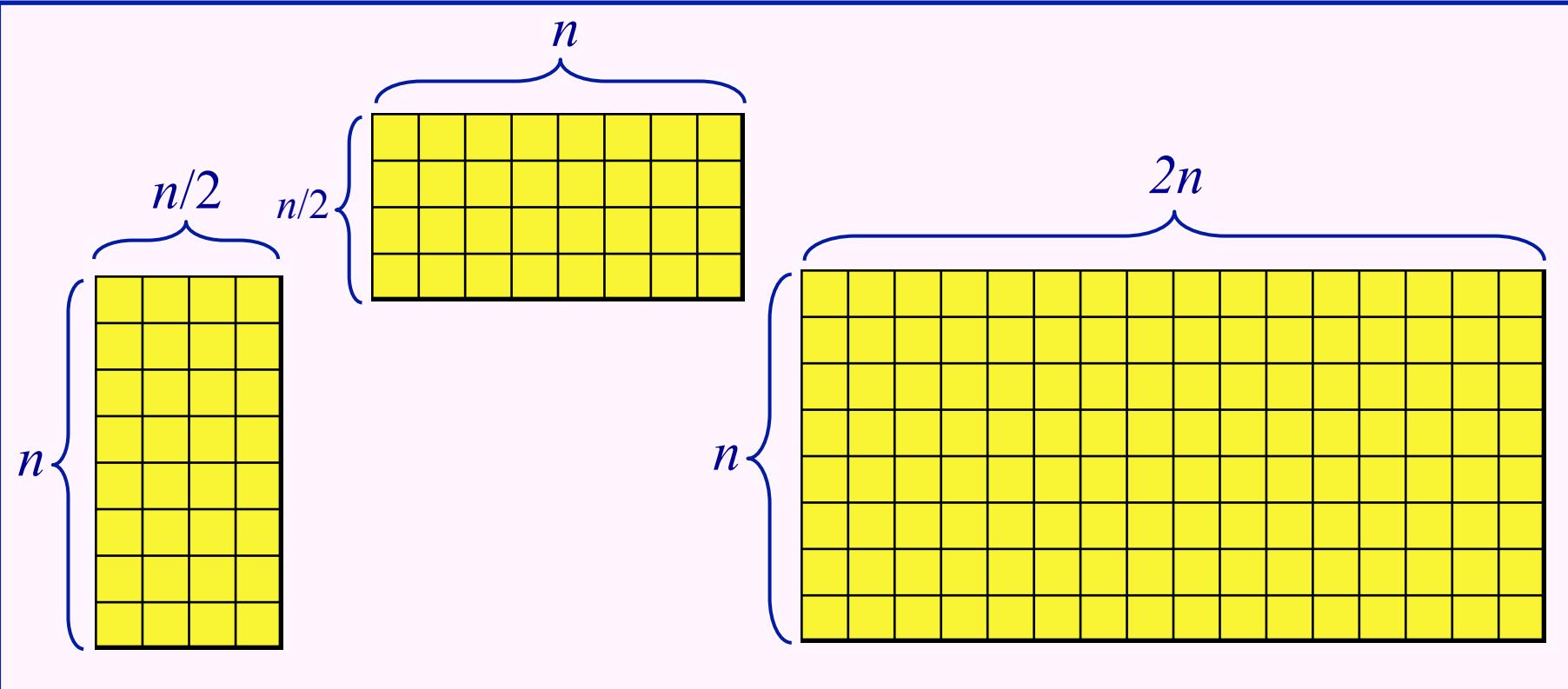
MATRIX-SUM (M, n, n) $\triangleright M[1..n; 1..n]$

```
Sumsf ← 0
for  $k \leftarrow 1$  to  $n$ 
    for  $j \leftarrow 1$  to  $n$ 
        do  $Sumsf \leftarrow Sumsf + M[k, j]$ 
    Sum ← Sumsf
```

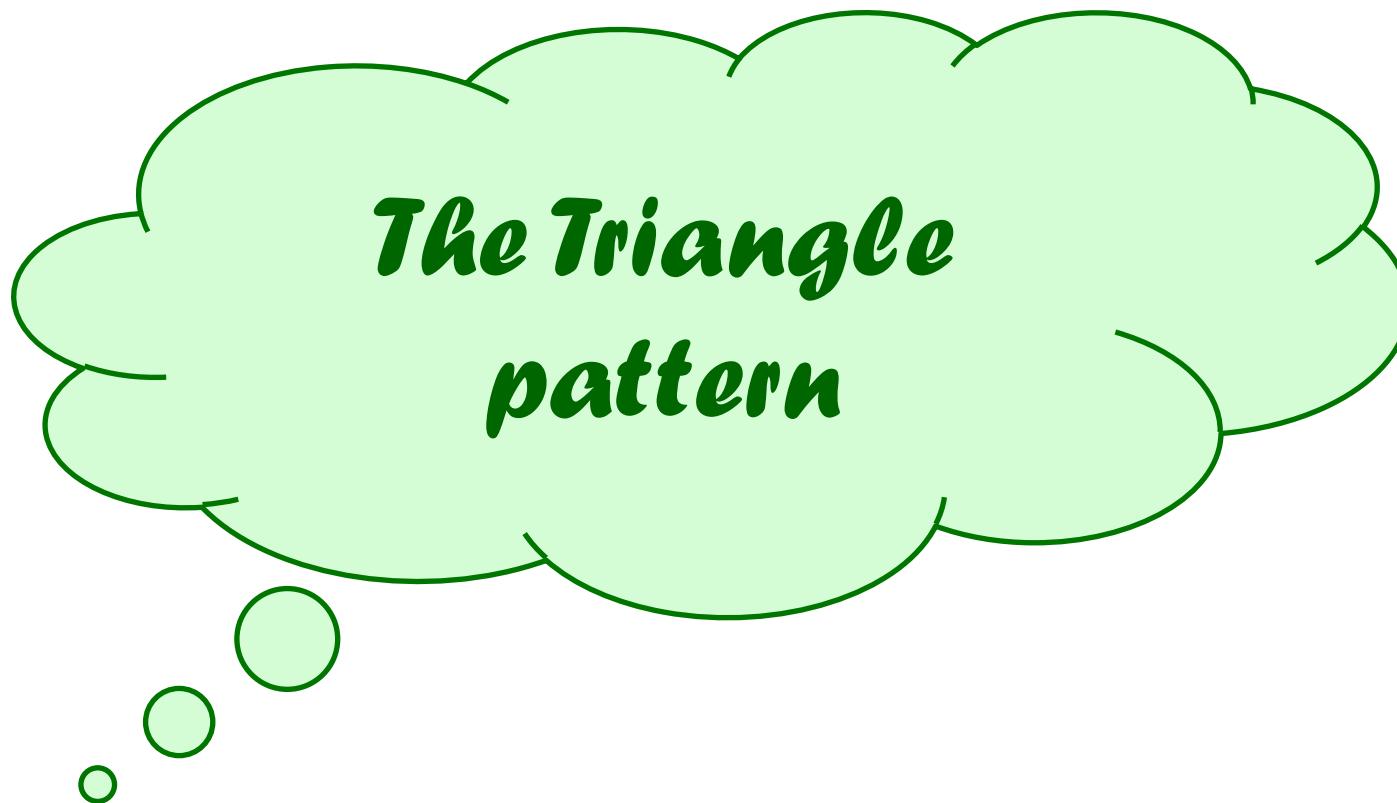
2D-Box Pattern (variants)

**AA-Pattern:
2D-Box**

$$T(n) = \sum_{k=1}^{dn} cn = \sum_{k=1}^{dn} \left(\sum_{j=1}^n c \right) = \Theta(n^2)$$



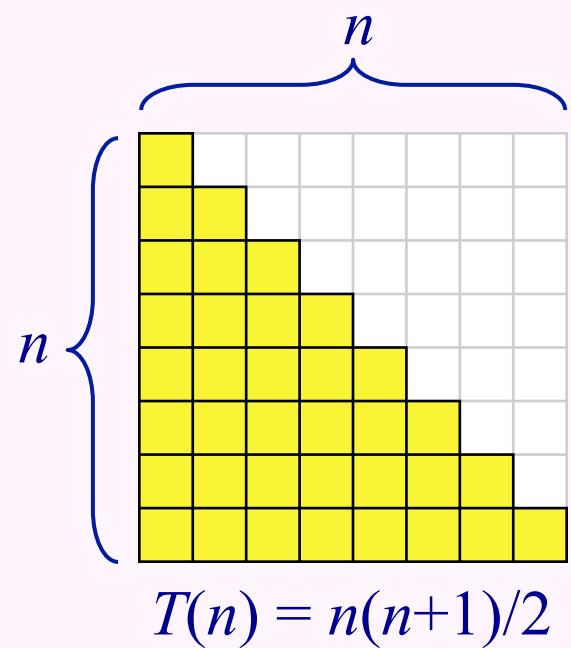
Algorithm-Analysis Pattern?



Algorithm Analysis Pattern

AA-Pattern: Triangle

$$T(n) = \sum_{k=1}^n ck = \sum_{k=1}^n \left(\sum_{j=1}^k c \right) = \Theta(n^2)$$



Code Pattern

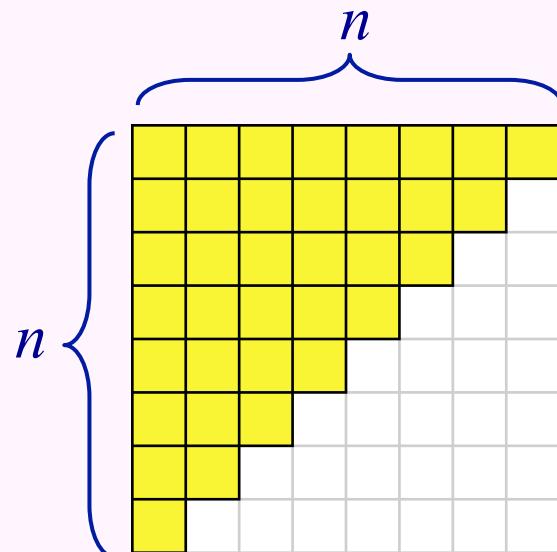
TRIANGLE-SUM (M, n, n) $\triangleright M[1..n; 1..n]$

```
Sumsf  $\leftarrow 0$ 
for  $k \leftarrow 1$  to  $n$ 
    for  $j \leftarrow 1$  to  $k$ 
        do  $Sumsf \leftarrow Sumsf + M[k, j]$ 
    Sum  $\leftarrow Sumsf$ 
```

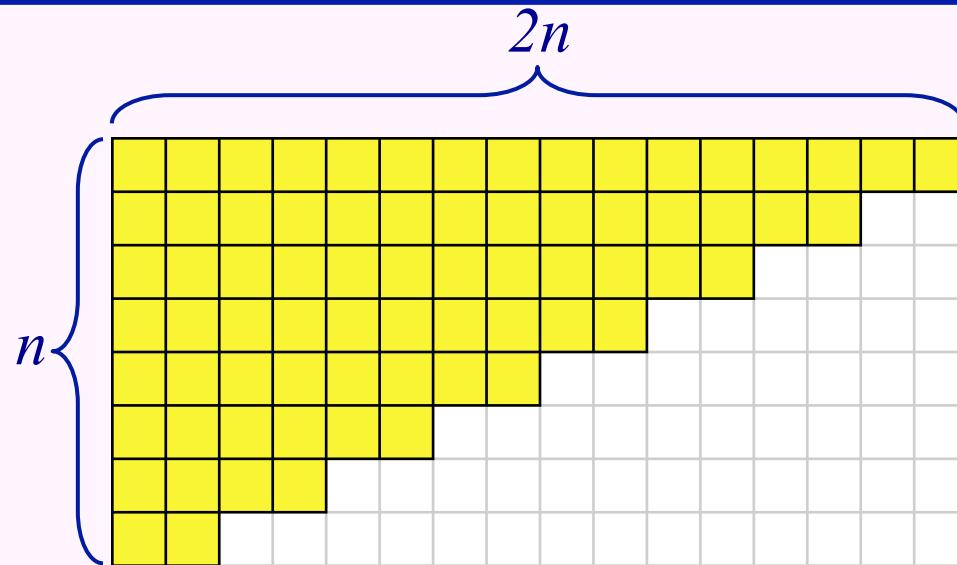
Triangle Pattern (variants)

AA-Pattern: Triangle

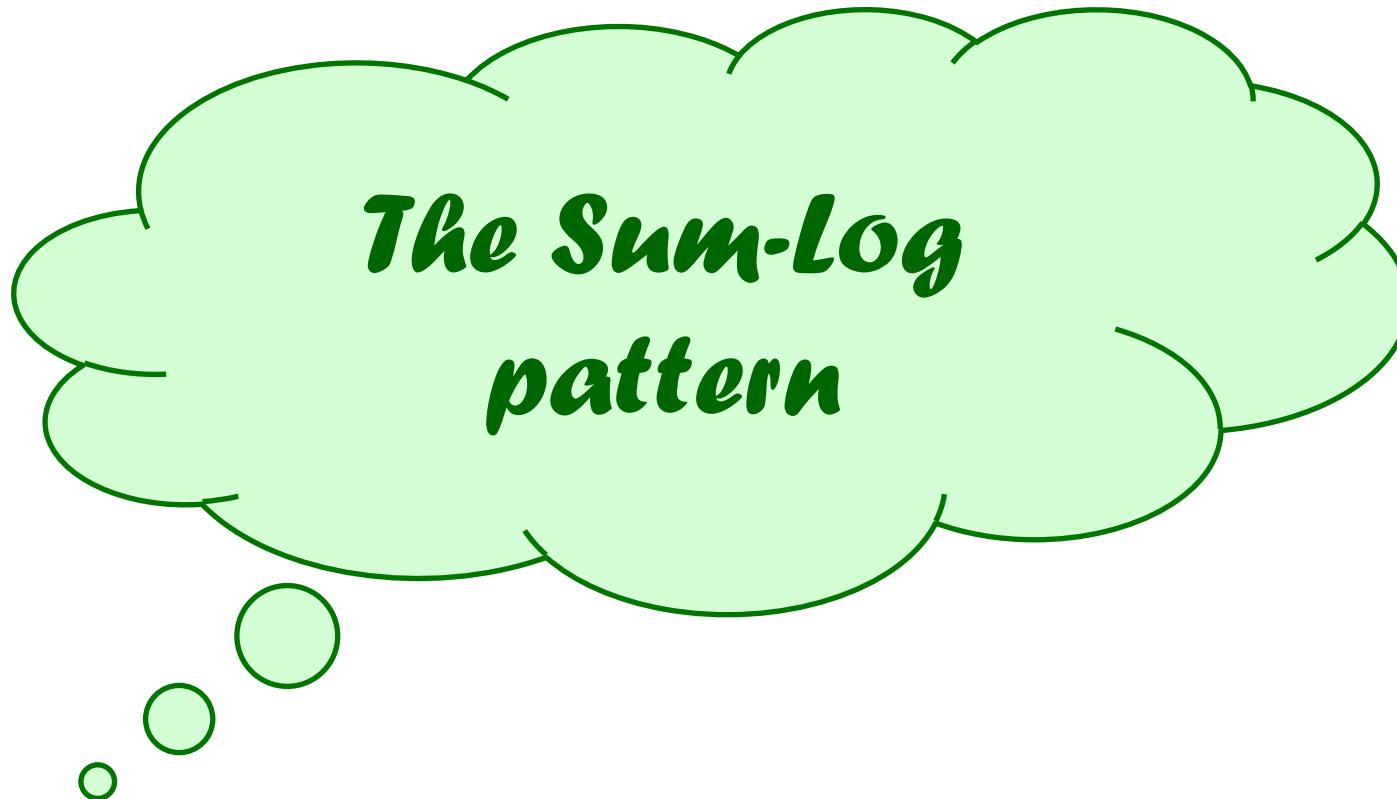
$$T(n) = \sum_{k=1}^n ck = \sum_{k=1}^n \left(\sum_{j=1}^k c \right) = \Theta(n^2)$$



$$T(n) = n(n+1)/2$$



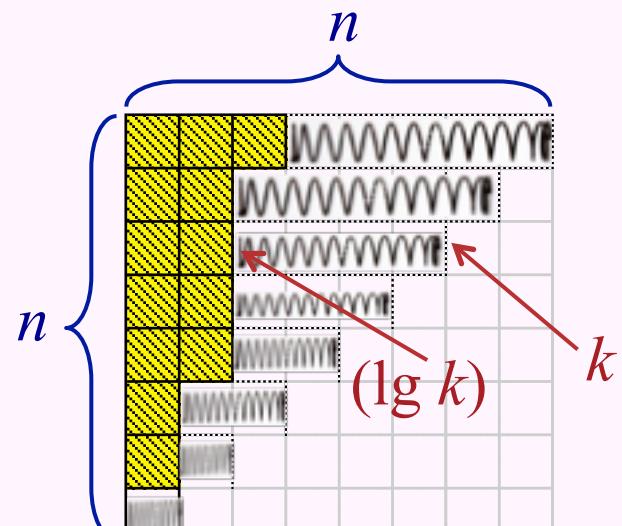
Algorithm-Analysis Pattern?



Sum-Log Pattern

AA-Pattern: Sum-Log

$$T(n) = \sum_{k=1}^n (\lg k) = \Theta(n \lg n)$$
$$= (\lg 1) + (\lg 2) + \dots + (\lg n)$$



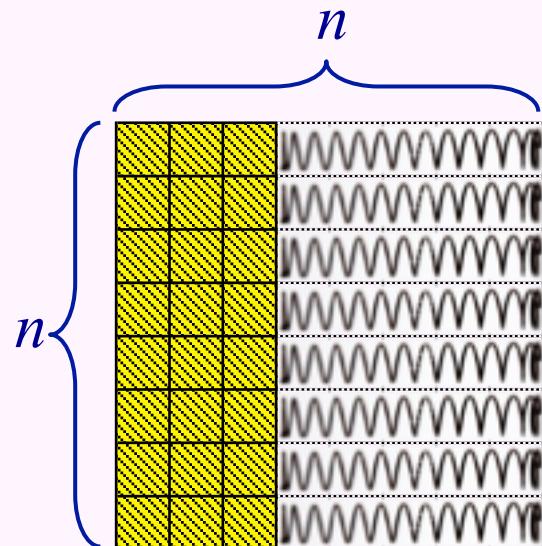
Code Pattern

```
SUM-LOG ( $A, n, X$ )  $\triangleright A[1..n]$   
(* bin-search  $A[1..k]$ ,  $k=1,2,\dots,n$  *)  
for  $k \leftarrow n$  downto 1  
    Binary-Search( $A, 1, k, X$ )
```

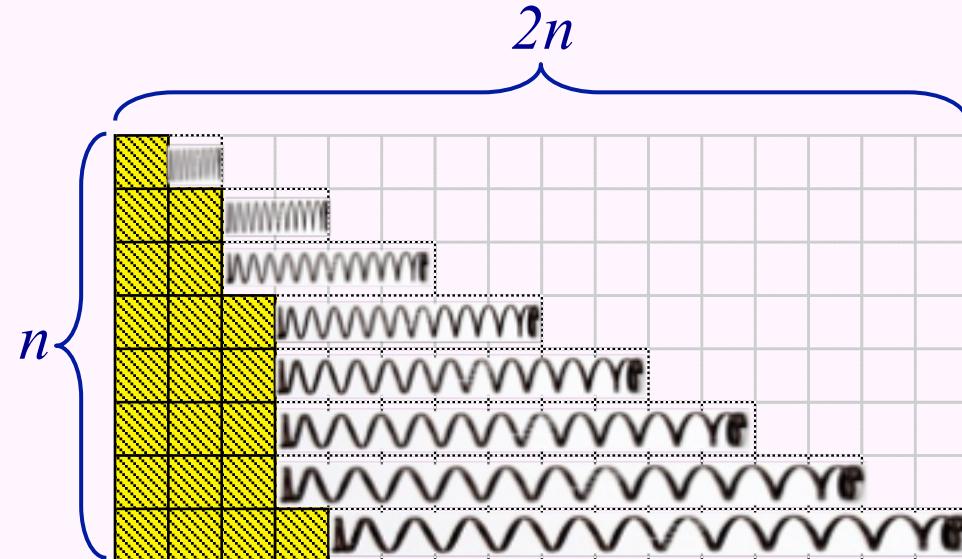
Sum-Log Pattern (variants)

**AA-Pattern:
Sum-Log**

$$T(n) = \sum_{k=1}^n (\lg k) = \Theta(n \lg n)$$
$$= (\lg 1) + (\lg 2) + \dots + (\lg n)$$

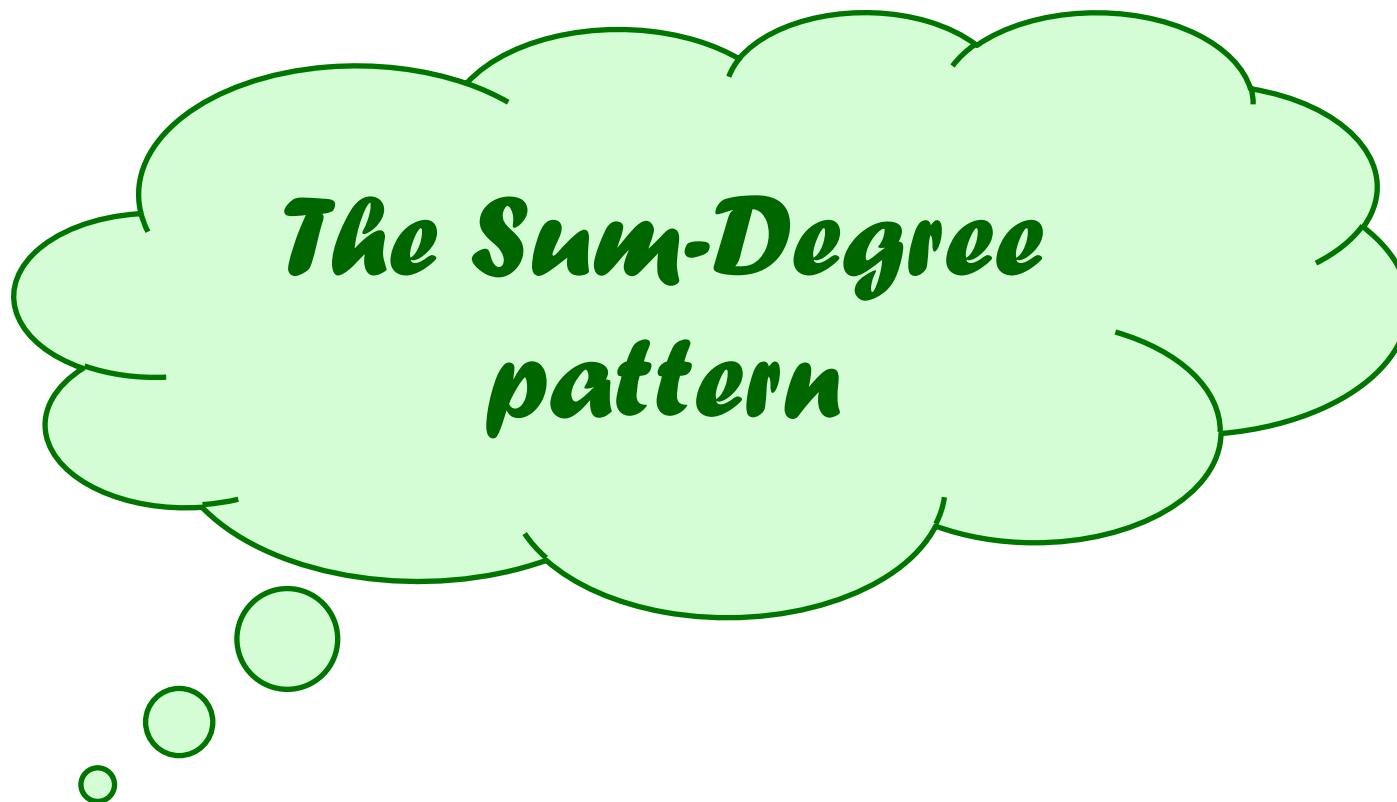


$$T(n) = \Theta(n \lg n)$$



$$T(n) = \Theta(n \lg n)$$

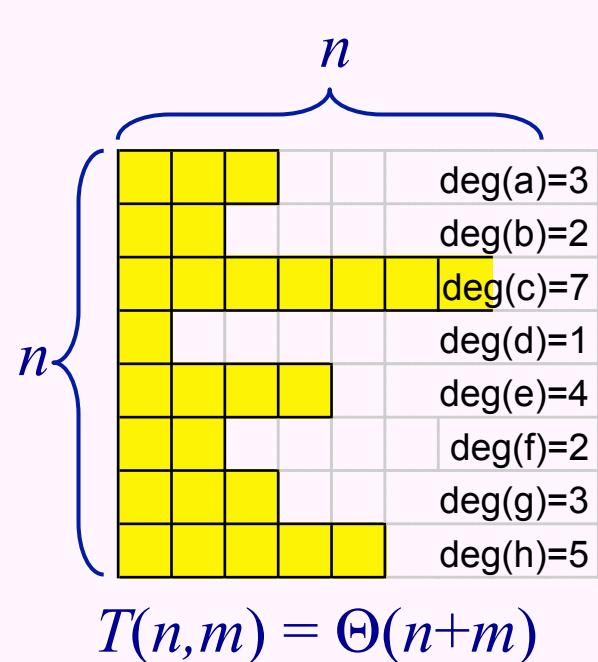
Algorithm-Analysis Pattern?



Sum-Degree Pattern

AA-Pattern: Sum-Degree

$$T(n,m) = \sum_{k=1}^n \deg(v_k) = 2m = \Theta(n + m)$$
$$= \deg(v_1) + \deg(v_2) + \dots + \deg(v_n)$$



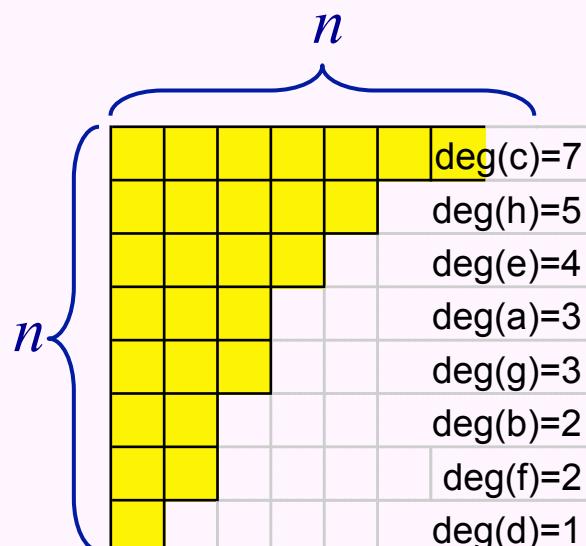
Code Pattern

SUM-DEGREE (G, n, m) $\triangleright G=(V,E)$
(* V has n nodes, E has m edges *)
for each v in G **do**
 for each w in $Adj(v)$
 Print details of (v, w)

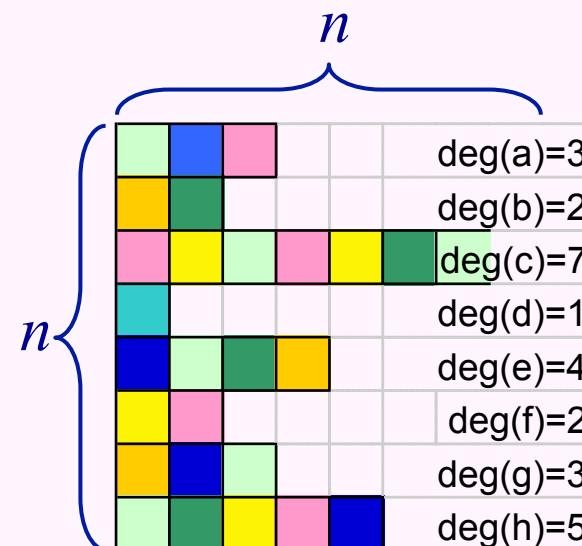
Sum-Degree Pattern (variants)

AA-Pattern: Sum-Degree

$$T(n,m) = \sum_{k=1}^n \deg(v_k) = 2m = \Theta(m)$$
$$= \deg(v_1) + \deg(v_2) + \dots + \deg(v_n)$$



$$T(n) = \Theta(n+m)$$



$$T(n) = \Theta(n+m)$$

HW for myself and students...

This is for CS3230 staff and students:

Anyone interested in volunteering to help in this,
please email me and say you like to help.

Thanks in advance.

- For each AA-pattern, list the different algorithms that has that AA-pattern.
- Do it for all the algorithms covered in CS1020, CS2010, CS2020.
- Discover new AA-patterns

HW for myself and students...

□ IDEA by HW:

- ❖ Create flash-card of each AA-pattern.
- ❖ From page has definition of pattern,
back page has variant pattern and
corresponding algorithms;

Looking for people who are good with artistic design to prototype some nice, cool, AA-pattern flash-cards

Thank you.

Q & A



School *of* Computing

Asymptotic Notation, Review of Functions & Summations

(Notes from Prof. David Plaisted,
UNC, Chapel Hill)

READ: [CLRS]-Ch.2-3, App-A

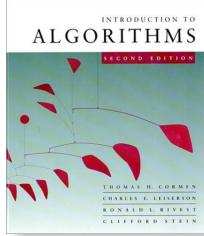
Heavily edited by HonWai LEONG, NUS

Asymptotic Complexity

- ◆ Running time of an algorithm as a function of input size n **for large n** .
- ◆ Expressed using only the **highest-order term** in the expression for the exact running time.
 - ◆ Instead of exact running time, say $\Theta(n^2)$.
 - ◆ Describes behavior of function in the limit.
 - ◆ Written using *Asymptotic Notation*.

Asymptotic Notation

- ◆ $\Theta, O, \Omega, o, \omega$
- ◆ Defined for functions over the natural numbers.
 - ◆ Ex: $f(n) = O(n^2)$.
 - ◆ Describes how $f(n)$ grows in comparison to n^2 .
- ◆ Define a *set* of functions; in practice used to compare two function sizes.
- ◆ The notations describe different *rate-of-growth relations* between the defining function and the defined set of functions.

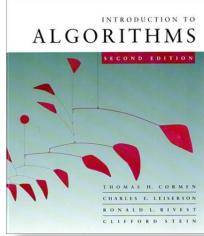


Asymptotic notation

O -notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

*We assume that $f(n)$ and $g(n)$ are nonnegative, for large n .
(After all they measure running times of algorithms.)*



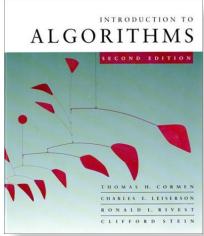
Asymptotic notation

O -notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

EXAMPLE: Prove that $2n^2 = O(n^3)$

Proof by defn: Need to find constants c , n_0
Start with $f(n) = 2n^2 \leq 2n^3$ for all $(n \geq 1)$.
So choose $c = 2$, $n_0 = 1$. Hence, $f(n) = O(n^3)$.



Many ways to prove this...

EXAMPLE: $2n^2 = O(n^3)$

Proof by defn-2: Need to find constants c, n_0

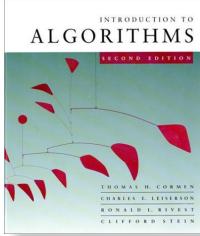
Start with $f(n) = 2n^2$

We note that $2n^2 \leq 1n^3$ for all $(n \geq 2)$.

So choose $c = 1, n_0 = 2$. Hence, $f(n) = O(n^3)$.

In general: Any method to find constants c, n_0
so that $f(n) = 2n^2 \leq c n^3$ for all $(n \geq n_0)$. (**)

Any c, n_0 satisfying (**) will do!



Asymptotic notation

O -notation (upper bounds):

We write $f(n) = O(g(n))$ if there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

EXAMPLE: $2n^2 = O(n^3)$ ($c = 2, n_0 = 1$)

*functions,
not values*



*funny, “one-way”
equality*

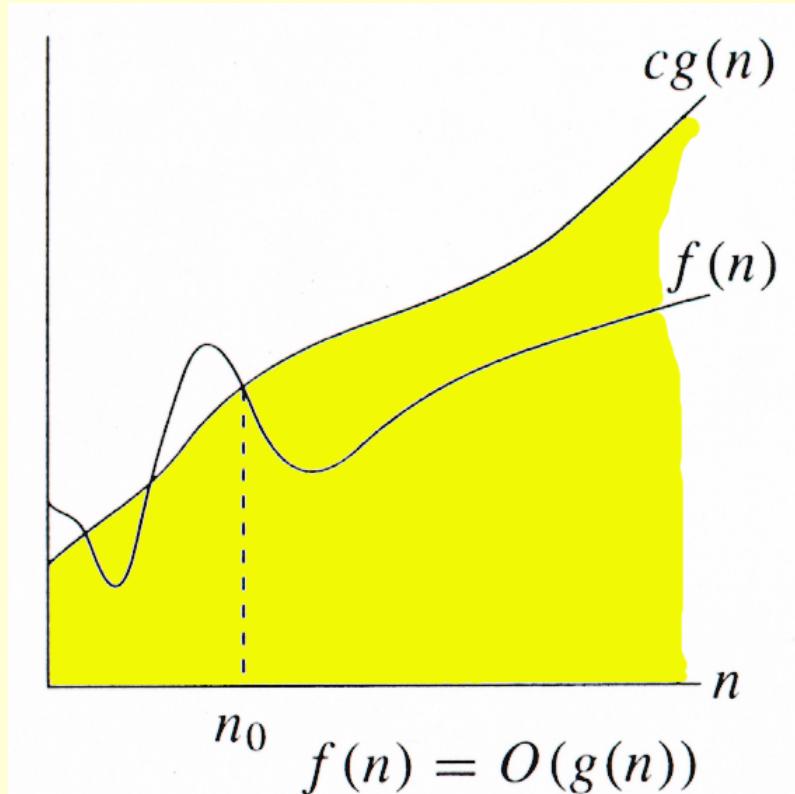
O-notation

For function $g(n)$, define $O(g(n))$, big-O of n , as the set:

$O(g(n)) = \{f(n) : \exists \text{ constants } c > 0 \text{ and } n_0 > 0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq f(n) \leq c g(n)\}$

Actually: $O(g(n))$ is set of all functions whose *rate of growth* is the *same as or lower than* that of $g(n)$.

$g(n)$ is an **asymptotic upper bound** for $f(n)$.



Technically, correct usage is $f(n) \in \Theta(g(n))$.

Informal usage, $f(n) = \Theta(g(n))$.

I'll accept either... (used in textbook, notes)

More Examples

$O(g(n)) = \{f(n) : \exists \text{ positive constants } c \text{ and } n_0,$
 $\text{such that } \forall n \geq n_0, \text{ we have } 0 \leq f(n) \leq cg(n) \}$

- ♦ $f(n) = 5n^2 + 20n$ is $O(n^2)$. How?

By defn: Need to find constants c, n_0 . Start with
 $f(n) = 5n^2 + 20n$ (use maths)
 $\leq 5n^2 + 20n^2$ for all $(n \geq 1)$ ($20n < 20n^2$)
 $\leq 25n^2$ for all $(n \geq 1)$

So choose $c = 25, n_0 = 1$. Hence, $f(n) = O(n^2)$.

Any quadratic function ($an^2 + bn + c$) is $O(n^2)$. DIY

(Here a has to be positive.)

More Examples

$O(g(n)) = \{f(n) : \exists \text{ positive constants } c \text{ and } n_0,$
 $\text{such that } \forall n \geq n_0, \text{ we have } 0 \leq f(n) \leq cg(n) \}$

♦ $f(n) = 7n^3 - 14n^2 + 12n = O(n^3)$

By defn: Need to find constants c, n_0 . Start with
 $f(n) = 7n^3 - 14n^2 + 12n$ (use maths)

$$\leq 7n^3 + 12n^3 \text{ for all } (n \geq 1) \quad (12n < 12n^3)$$

$$\leq 19n^3 \text{ for all } (n \geq 1)$$

So choose $c = 19, n_0 = 1$. Hence, $f(n) = O(n^3)$.

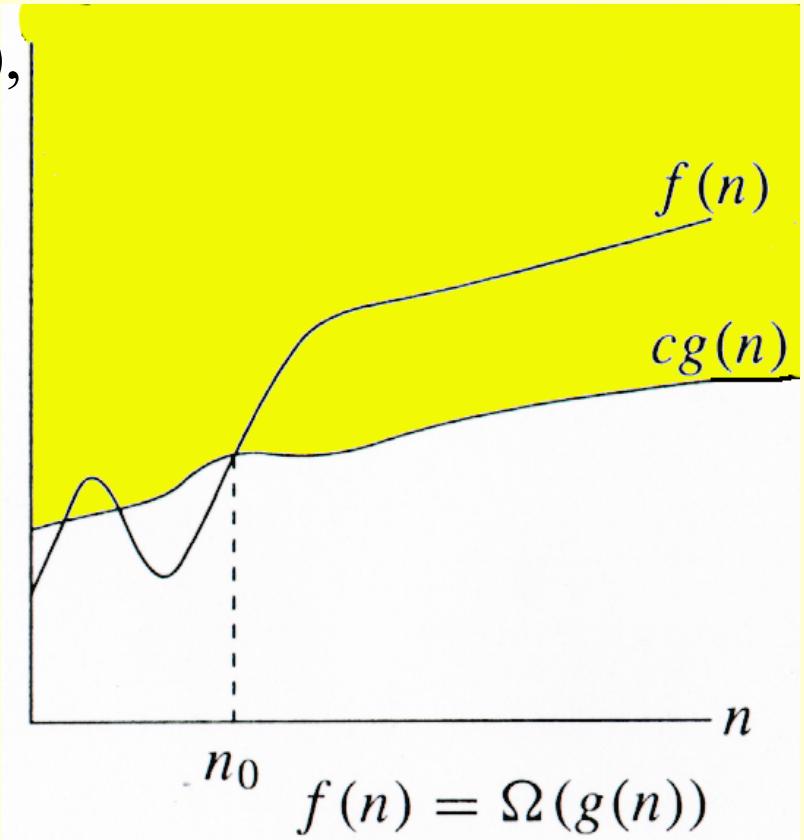
Any polynomial function of degree k is $O(n^k)$.

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n , as the set:

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0 ,
such that $\forall n \geq n_0$,
we have $0 \leq c g(n) \leq f(n)\}$

Intuitively: Set of all functions
whose *rate of growth* is the *same*
as or higher than that of $g(n)$.



$g(n)$ is an *asymptotic lower bound* for $f(n)$.

Example

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c and n_0 , such that $\forall n \geq n_0$, we have $0 \leq c g(n) \leq f(n)\}$

- ♦ $f(n) = 2n^2 + 20n$ is $O(n)$. How?

By defn: Need to find constants c, n_0 . Start with
 $f(n) = 2n^2 + 20n$ (use maths)

$\geq 20n$ for all $(n \geq 1)$ (Throw away $2n^2$)

So choose $c = 20, n_0 = 1$. Hence, $f(n) = \Omega(n)$.

More Examples

$\Omega(g(n)) = \{f(n) : \exists \text{ positive constants } c \text{ and } n_0, \text{ such that } \forall n \geq n_0, \text{ we have } 0 \leq c g(n) \leq f(n)\}$

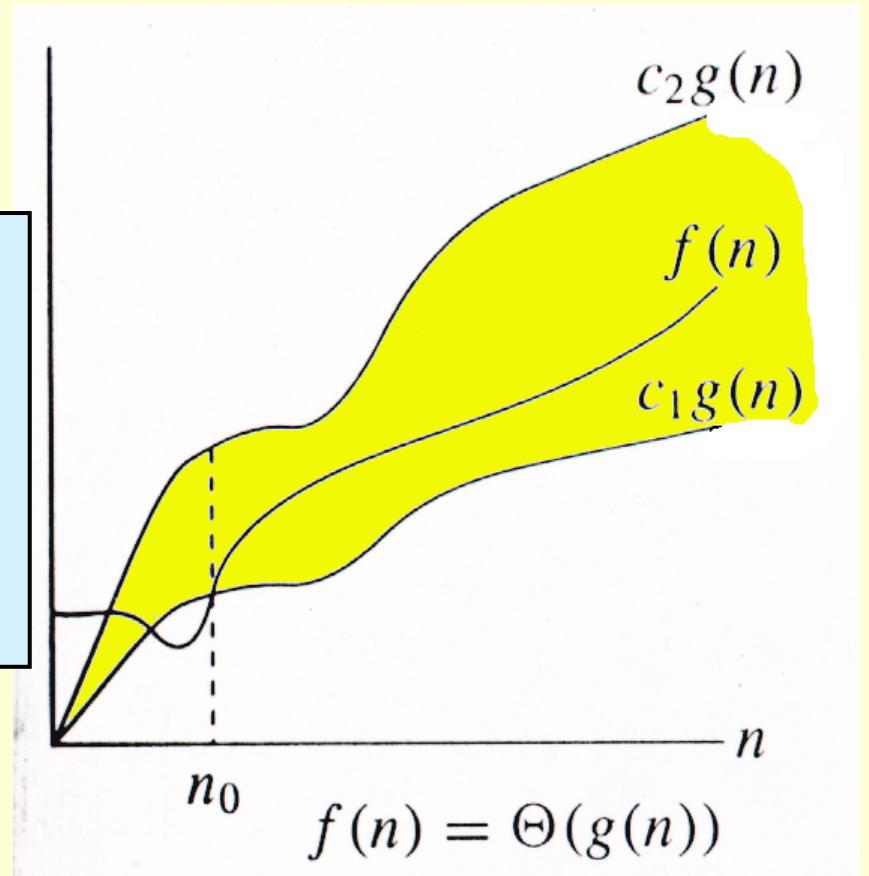
$$f(n) = 7n^3 - 14n^2 + 12n = \Omega(n^4)$$

By defn: Need to find constants c, n_0 . Start with
 $f(n) = 7n^3 - 14n^2 + 12n$ (use maths)
 $\geq 7n^3 - 14n^2$ (Throw away $12n$)
 $\geq 6n^3 + (n^3 - 14n^2)$ (split up $7n^3$)
 $\geq 6n^3$ for $n \geq 14$ ($(n^3 - 14n^2) > 0$)

So choose $c = 6, n_0 = 14$. Hence, $f(n) = \Omega(n^3)$.

Θ -notation

For function $g(n)$, define $\Theta(g(n))$, big-Theta of $g(n)$, as the set:

$$\begin{aligned}\Theta(g(n)) = & \{f(n) : \exists \text{ positive constants } c_1, c_2, \text{ and} \\ & n_0, \text{ such that } \forall n \geq n_0, \\ & \text{we have } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}.\end{aligned}$$


Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an **asymptotically tight bound** for $f(n)$.

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)).$$

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n)).$$

Example

$\Theta(g(n)) = \{f(n) : \exists$ positive constants c_1, c_2 , and n_0 ,
such that $\forall n \geq n_0, 0 \leq c_1g(n) \leq f(n) \leq c_2g(n)\}$

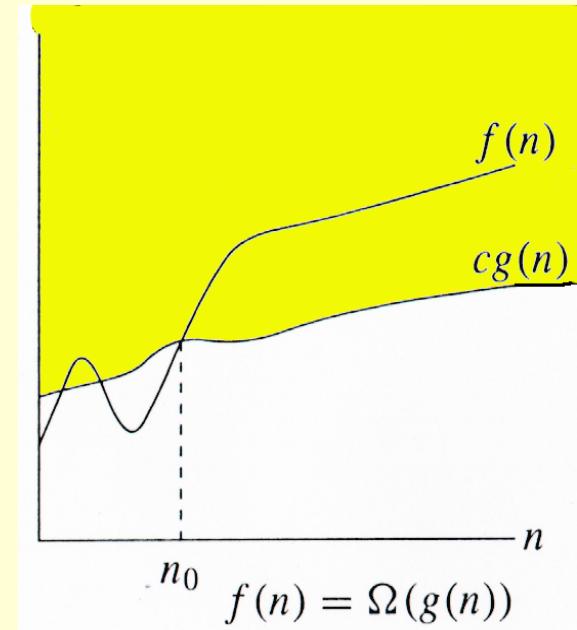
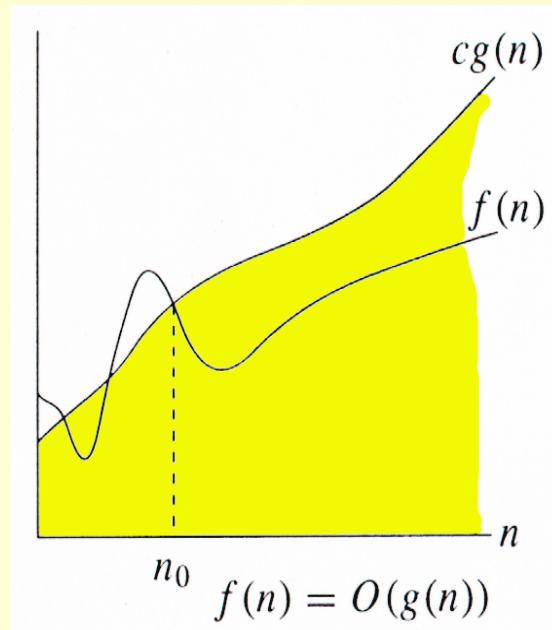
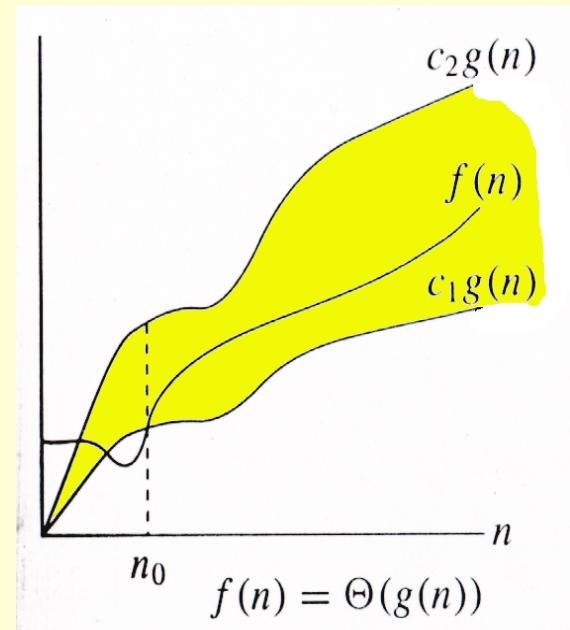
- ◆ $10n^2 - 3n = \Theta(n^2)$
- ◆ What constants for n_0, c_1 , and c_2 will work?
- ◆ Make c_1 a little smaller than the leading coefficient, and c_2 a little bigger.
- ◆ *To compare orders of growth, look at the leading term.*
- ◆ Exercise: Prove that $n^2/2 - 3n = \Theta(n^2)$

Example

$\Theta(g(n)) = \{f(n) : \exists \text{ positive constants } c_1, c_2, \text{ and } n_0,$
 $\text{such that } \forall n \geq n_0, \quad 0 \leq c_1g(n) \leq f(n) \leq c_2g(n)\}$

- ◆ Is $3n^3 \in \Theta(n^4)$??
- ◆ How about $2^{2n} \in \Theta(2^n)$??

Relations Between Θ , O , Ω



Relations Between Θ , Ω , O

Theorem : For any two functions $g(n)$ and $f(n)$,

$$f(n) = \Theta(g(n)) \text{ iff}$$

$$f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n)).$$

- ◆ i.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- ◆ In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

Running Times

- ◆ “Running time is $O(f(n))$ ” \Rightarrow Worst case is $O(f(n))$
- ◆ $O(f(n))$ bound on the worst-case running time \Rightarrow $O(f(n))$ bound on the running time of every input.
- ◆ $\Theta(f(n))$ bound on the worst-case running time $\not\Rightarrow$ $\Theta(f(n))$ bound on the running time of every input.
- ◆ “Running time is $\Omega(f(n))$ ” \Rightarrow Best case is $\Omega(f(n))$
- ◆ Can still say “worst-case running time is $\Omega(f(n))$ ”
 - ◆ Means worst-case running time is given by some unspecified function $g(n) \in \Omega(f(n))$.

Example

- ◆ *Insertion sort* takes $\Theta(n^2)$ in the worst case, so sorting (as a *problem*) is $O(n^2)$.
- ◆ Any sort algorithm must look at each item, so sorting is $\Omega(n)$.
- ◆ In fact, using (e.g.) merge sort, sorting is $\Theta(n \lg n)$ in the worst case.
 - ◆ Later, we will prove that we cannot hope that any comparison sort to do better in the worst case.

Asymptotic Notation in Equations

- ◆ Can use asymptotic notation in equations to replace expressions containing lower-order terms.
- ◆ For example,

$$\begin{aligned}4n^3 + 3n^2 + 2n + 1 &= 4n^3 + 3n^2 + \Theta(n) \\&= 4n^3 + \Theta(n^2) = \Theta(n^3).\end{aligned}$$

How to interpret?

- ◆ In equations, $\Theta(f(n))$ always stands for an ***anonymous function*** $g(n) \in \Theta(f(n))$
 - ◆ In the example above, $\Theta(n^2)$ stands for $3n^2 + 2n + 1$.

o-notation (little-*o*)

For a given function $g(n)$, the set little-*o*:

$$o(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \text{ such that} \\ \forall n \geq n_0, \text{ we have } 0 \leq f(n) < cg(n)\}.$$

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \rightarrow \infty} [f(n) / g(n)] = 0$$

$g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Observe the difference in this definition from previous ones. **Why?**

ω -notation

For a given function $g(n)$, the set little-omega:

$$\Omega(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \text{ such that} \\ \forall n \geq n_0, \text{ we have } 0 \leq cg(n) < f(n)\}.$$

$f(n)$ becomes arbitrarily large relative to $g(n)$ as n approaches infinity:

$$\lim_{n \rightarrow \infty} [f(n) / g(n)] = \infty.$$

$g(n)$ is a ***lower bound*** for $f(n)$ that is not asymptotically tight.

Comparison of Functions

$$f \Leftrightarrow g \approx a \Leftrightarrow b$$

$$f(n) = O(g(n)) \approx a \leq b$$

$$f(n) = \Omega(g(n)) \approx a \geq b$$

$$f(n) = \Theta(g(n)) \approx a = b$$

$$f(n) = o(g(n)) \approx a < b$$

$$f(n) = \omega(g(n)) \approx a > b$$

Limits

- ◆ $\lim_{n \rightarrow \infty} [f(n) / g(n)] = 0$ $\Rightarrow f(n) \in o(g(n))$
- ◆ $\lim_{n \rightarrow \infty} [f(n) / g(n)] < \infty$ $\Rightarrow f(n) \in O(g(n))$
- ◆ $0 < \lim_{n \rightarrow \infty} [f(n) / g(n)] < \infty$ $\Rightarrow f(n) \in \Theta(g(n))$
- ◆ $0 < \lim_{n \rightarrow \infty} [f(n) / g(n)]$ $\Rightarrow f(n) \in \Omega(g(n))$
- ◆ $\lim_{n \rightarrow \infty} [f(n) / g(n)] = \infty$ $\Rightarrow f(n) \in \omega(g(n))$
- ◆ $\lim_{n \rightarrow \infty} [f(n) / g(n)]$ undefined \Rightarrow can't say

Properties

◆ Transitivity

$$f(n) = \Theta(g(n)) \text{ & } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \text{ & } g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ & } g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \text{ & } g(n) = o(h(n)) \Rightarrow f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \text{ & } g(n) = \omega(h(n)) \Rightarrow f(n) = \omega(h(n))$$

◆ Reflexivity

$$f(n) = \Theta(f(n))$$

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

Properties

- ◆ **Symmetry**

$$f(n) = \Theta(g(n)) \text{ iff } g(n) = \Theta(f(n))$$

- ◆ **Complementarity**

$$f(n) = O(g(n)) \text{ iff } g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \text{ iff } g(n) = \omega(f(n))$$

Common Functions

Monotonicity

- ♦ $f(n)$ is
 - ♦ **monotonically increasing** if $m \leq n \Rightarrow f(m) \leq f(n)$.
 - ♦ **monotonically decreasing** if $m \leq n \Rightarrow f(m) \geq f(n)$.
 - ♦ **strictly increasing** if $m < n \Rightarrow f(m) < f(n)$.
 - ♦ **strictly decreasing** if $m < n \Rightarrow f(m) > f(n)$.

Exponentials

- ◆ Useful Identities:

$$a^{-1} = \frac{1}{a}$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

- ◆ Exponentials and polynomials

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

$$\Rightarrow n^b = o(a^n)$$

Logarithms

$x = \log_b a$ is the exponent for $a = b^x$.

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

Natural log: $\ln a = \log_e a$

$$\log_c(ab) = \log_c a + \log_c b$$

Binary log: $\lg a = \log_2 a$

$$\log_b a^n = n \log_b a$$

$$\lg^2 a = (\lg a)^2$$

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

$$\lg \lg a = \lg(\lg a)$$

$$\log_b(1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

Logarithms and exponentials – Bases

- ♦ If the base of a logarithm is changed from one constant to another, the value is altered by a constant factor.
 - ♦ Ex: $\log_{10} n * \log_2 10 = \log_2 n.$
 - ♦ Base of logarithm is not an issue in asymptotic notation.
- ♦ Exponentials with different bases differ by a exponential factor (not a constant factor).
 - ♦ Ex: $2^n = (2/3)^n * 3^n.$

Polylogarithms

- ♦ For $a \geq 0, b > 0$, $\lim_{n \rightarrow \infty} (\lg^a n / n^b) = 0$,
so $\lg^a n = o(n^b)$, and $n^b = \omega(\lg^a n)$
 - ♦ Prove using L'Hopital's rule repeatedly
- ♦ $\lg(n!) = \Theta(n \lg n)$
 - ♦ Prove using Stirling's approximation (in the text) for $\lg(n!)$.

Exercise

Express functions in A in asymptotic notation using functions in B.

A

$$5n^2 + 100n$$

B

$$3n^2 + 2$$

$$A = \Theta(B)$$

$$A \in \Theta(n^2), n^2 \in \Theta(B) \Rightarrow A \in \Theta(B)$$

$$\log_3(n^2)$$

$$\log_2(n^3)$$

$$A = \Theta(B)$$

$$\log_b a = \log_c a / \log_c b; A = 2\lg n / \lg 3, B = 3\lg n, A/B = 2/(3\lg 3)$$

$$n^{\lg 4}$$

$$3^{\lg n}$$

$$A = \omega(B)$$

$$a^{\log b} = b^{\log a}; B = 3^{\lg n} = n^{\lg 3}; A/B = n^{\lg(4/3)} \rightarrow \infty \text{ as } n \rightarrow \infty$$

$$\lg^2 n$$

$$n^{1/2}$$

$$A = o(B)$$

$$\lim_{n \rightarrow \infty} (\lg^a n / n^b) = 0 \text{ (here } a = 2 \text{ and } b = 1/2\text{)} \Rightarrow A \in o(B)$$

Summations – Review

Review on Summations

- ◆ Why do we need summation formulas?

For computing the running times of iterative constructs (loops). (CLRS – Appendix A)

Example: Maximum Subvector

Given an array $A[1 \dots n]$ of numeric values (can be positive, zero, and negative) determine the subvector $A[i \dots j]$ ($1 \leq i \leq j \leq n$) whose sum of elements is maximum over all subvectors.

1	-2	2	2
---	----	---	---

Review on Summations

```
MaxSubvector( $A$ ,  $n$ )
```

```
     $maxsum \leftarrow 0;$ 
    for  $i \leftarrow 1$  to  $n$  do
        for  $j = i$  to  $n$  do
             $sum \leftarrow 0$ 
            for  $k \leftarrow i$  to  $j$  do
                 $sum += A[k]$ 
             $maxsum \leftarrow \max(sum, maxsum)$ 
    return  $maxsum$ 
```

$$\blacklozenge T(n) = \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j \Theta(1) =$$

♦ NOTE: This is not a simplified solution. What *is* the final answer?

Review on Summations

- ◆ **Constant Series:** For integers a and b , $a \leq b$,

$$\sum_{i=a}^b 1 = b - a + 1$$

- ◆ **Linear Series (Arithmetic Series):** For $n \geq 0$,

$$\sum_{i=1}^n i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

- ◆ **Quadratic Series:** For $n \geq 0$,

$$\sum_{i=1}^n i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Review on Summations

- ◆ **Cubic Series:** For $n \geq 0$,

$$\sum_{i=1}^n i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

- ◆ **General Case:** For $n \geq 0$,

$$\sum_{k=1}^n k^s = \frac{1}{(s+1)} n^{s+1} + \Theta(n^s) = \Theta(n^{s+1})$$

Review on Summations

- ◆ **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \cdots + x^n = \frac{(x^{n+1} - 1)}{(x - 1)}$$

$$\text{For } |x| < 1, \quad \sum_{k=0}^{\infty} x^k = \frac{1}{(1 - x)}$$

Review on Summations

- ◆ **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

$$\sum_{k=1}^n kc^k = c + 2c^2 + \cdots + nc^n = \frac{nc^{n+2} - (n+1)c^{n+1} + c}{(c-1)^2}$$

- ◆ **Harmonic Series:** n^{th} harmonic number, $n \in \mathbb{I}^+$,

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \\ &= \sum_{k=1}^n \frac{1}{k} = \ln(n) + O(1) \end{aligned}$$

Review on Summations

- ◆ **Telescoping Series:**

$$\sum_{k=1}^n a_k - a_{k-1} = a_n - a_0$$

- ◆ **Differentiating Series:** For $|x| < 1$,

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Review on Summations

◆ **Approximation by integrals:**

- ♦ For monotonically increasing $f(n)$

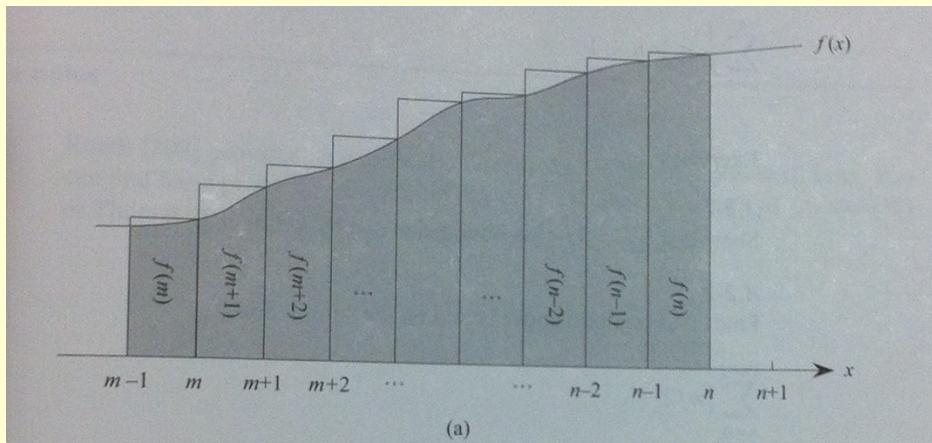
$$\int_{m-1}^n f(x)dx \leq \sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x)dx$$

- ♦ For monotonically decreasing $f(n)$

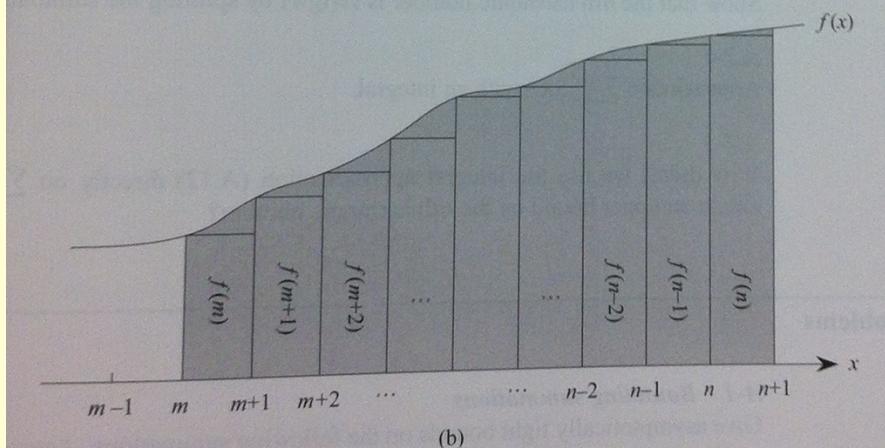
$$\int_m^{n+1} f(x)dx \leq \sum_{k=m}^n f(k) \leq \int_{m-1}^n f(x)dx$$

◆ **How?**

For Monotone Increasing Function



(a)



(b)

Figure A.1 Approximation of $\sum_{k=m}^n f(k)$ by integrals. The area of each rectangle is shown within the rectangle, and the total rectangle area represents the value of the summation. The integral is represented by the shaded area under the curve. By comparing areas in (a), we get $\int_{m-1}^n f(x) dx \leq \sum_{k=m}^n f(k)$, and then by shifting the rectangles one unit to the right, we get $\sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x) dx$ in (b).

$$\int_{m-1}^n f(x) dx \leq \sum_{k=m}^n f(k)$$

$$\sum_{k=m}^n f(k) \leq \int_m^{n+1} f(x) dx$$

Eg of Summation by Integration

♦ *n*th harmonic number

$$\ln(n+1) \leq \sum_{k=1}^n \frac{1}{k} \leq (\ln n) + 1$$

$$\begin{aligned}\sum_{k=1}^n \frac{1}{k} &\geq \int_1^{n+1} \frac{1}{x} dx = [\ln x]_1^{n+1} \\&= \ln(n+1)\end{aligned}$$

$$\begin{aligned}\sum_{k=2}^n \frac{1}{k} &\leq \int_1^n \frac{1}{x} dx = [\ln x]_1^n = (\ln n) \\&\Rightarrow \sum_{k=1}^n \frac{1}{k} \leq (\ln n) + 1\end{aligned}$$

Reading Assignment

- ◆ Chapter 2-3, App-A of [CLRS].