COMP319 Algorithms
Lecture 4
Asymptotic Performance Analysis

Instructor: Gil-Jin Jang

Asymptotic performance analysis

Big-O notation

Analyses of simple sorting algorithms

Master theorem

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

In this course, we care most about *asymptotic performance*

ASYMPTOTIC PERFORMANCE

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Asymptotic Performance

- How does algorithm behave as the problem size gets VERY LARGE?

- In terms of:
  - Running time
  - Memory/storage requirements

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Machine-Independent time

- **The RAM Model**
  - Machine independent algorithm design depends on a hypothetical computer called **Random Acces Machine (RAM)**.
- Assumptions: generic *uniprocessor* RAM
  - No concurrent (parallel) operations
  - Each simple operation such as +, -, *, /, if, etc takes exactly one time step.
  - Loops and subroutines (functions) are not considered simple operations.
  - Each memory acces takes exactly one time step (equally expensive to access)
  - Constant word (i.e. integer) size
    - Unless we are explicitly manipulating bits

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call, most statements roughly require the same amount of time

    - y = m * x + b → 2 basic operations
    - c = 5 / 9 * (t - 32) → 3 basic operations
    - z = f(x) + g(y) → 1 basic operation + 2 function calls

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Input Size

- defined in terms of functions whose domains are the set of ***natural numbers***
  - n = {1, 2, …}
- How we characterize input size depends on:
  - Number of items (sorting)
  - number of nodes & edges (graph algorithms)
- Time and space complexity
  - This is generally a function of the input size: "n"

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Kinds of analyses

Worst-case: *(usually)*

- *T(n) = maximum time of algorithm on any input of size n.*
- *Provides an upper bound on running time*
- *Example: "how much time AT LEAST is required to sort millions of integers"*

Average-case: *(sometimes)*

- *T(n) = expected time of algorithm over all inputs of size n.*
- *Provides the expected running time*
- *Need assumption of statistical distribution of inputs.*

Best-case: *(luckily)*

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

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Order of Growth

- Simplifications
  - Ignore actual and abstract statement costs
  - *Order of growth* is the interesting measure:
- Asymptotic (big-O) notation:
  - Read O as "Big-O" (you'll also hear it as "order")
    - Sometimes O($f(n)$) is regarded as a set of functions whose asymptotic behavior is same as $f(n)$ (e.g. $100n \in O(n)$)

$$O\left(n^2\right) = O\left(100n^2\right) = O\left(1000000n^2\right) \propto n^2$$

$$O\left(an^2 + bn\right) = O\left(an^2\right) \propto n^2$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

O-notation: Upper Bound

- For a given function $g(n)$, we denote by $O(g(n))$ the set of functions
  - $f(n)$ is a member of $O(g(n))$ if there exist positive constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
    - there existes some constant c s.t. always $f(n) \leq c \cdot g(n)$ for large enough n.
  - Formally
    - $O(g(n)) = \{ f(n)$: $\exists$ positive constants c and n_0 such that $f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0 \}$
- We use O-notation to give an asymptotic upper bound of a function, to within a constant factor.

Why Asymptotic Behavior?

- *What does* O(*n*) *running time mean?* O(n^2)*?* O(*n* log *n*)*?*
- *How does asymptotic running time relate to asymptotic memory usage?*

$$\lim_{n\to\infty} \frac{n^2}{n} = \infty$$

$$\lim_{n\to\infty} \frac{\log n}{n} = 0$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Big O Fact

- A polynomial of degree k is $O(n^k)$

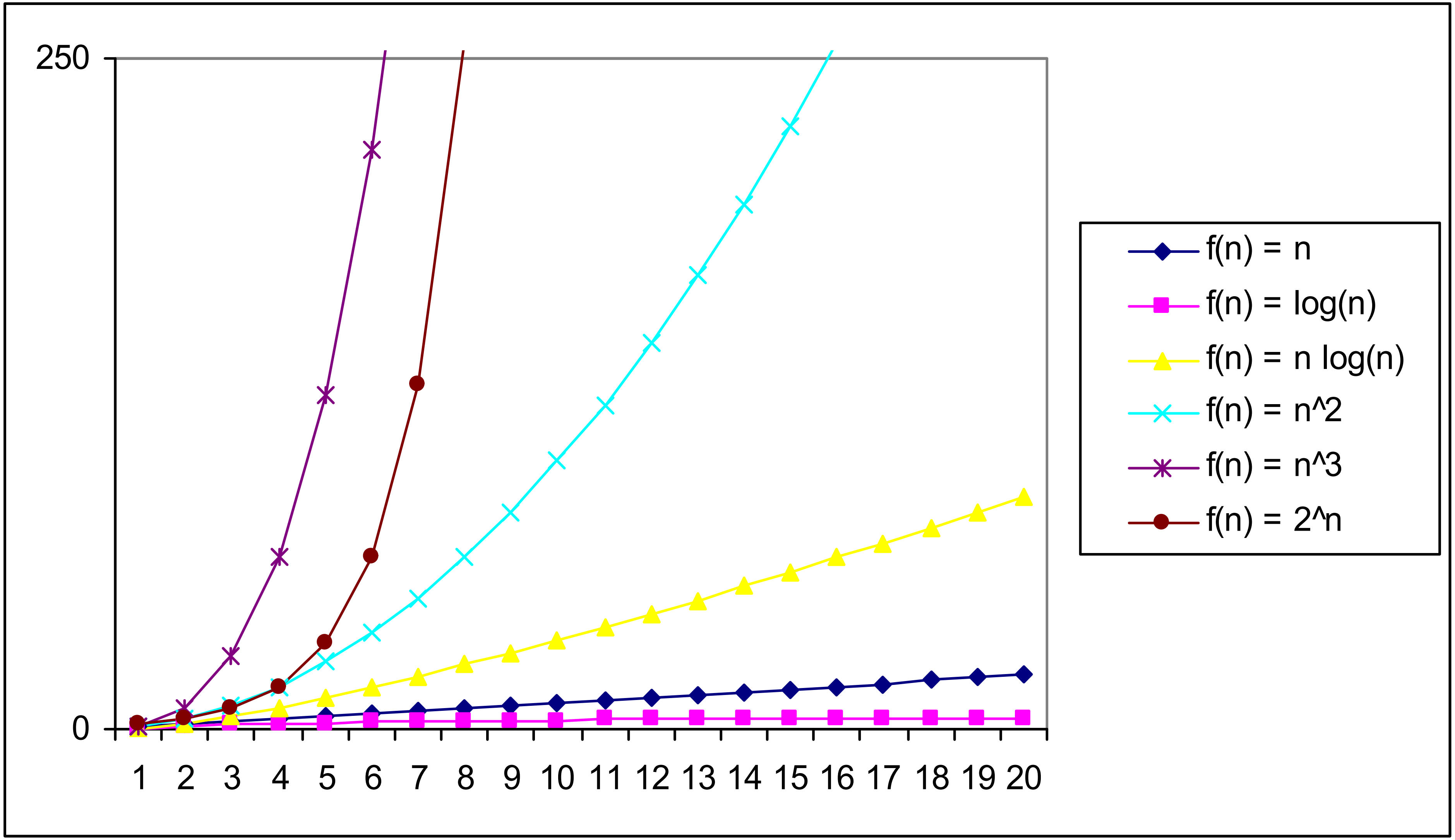
Proof:

- Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$
  - Let $a_i = | \, b_i \, |$
- $f(n) \leq a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

$$\leq \quad n^k \sum a_i \frac{n^i}{n^k} \quad \leq \quad n^k \sum a_i \quad \leq \quad cn^k$$

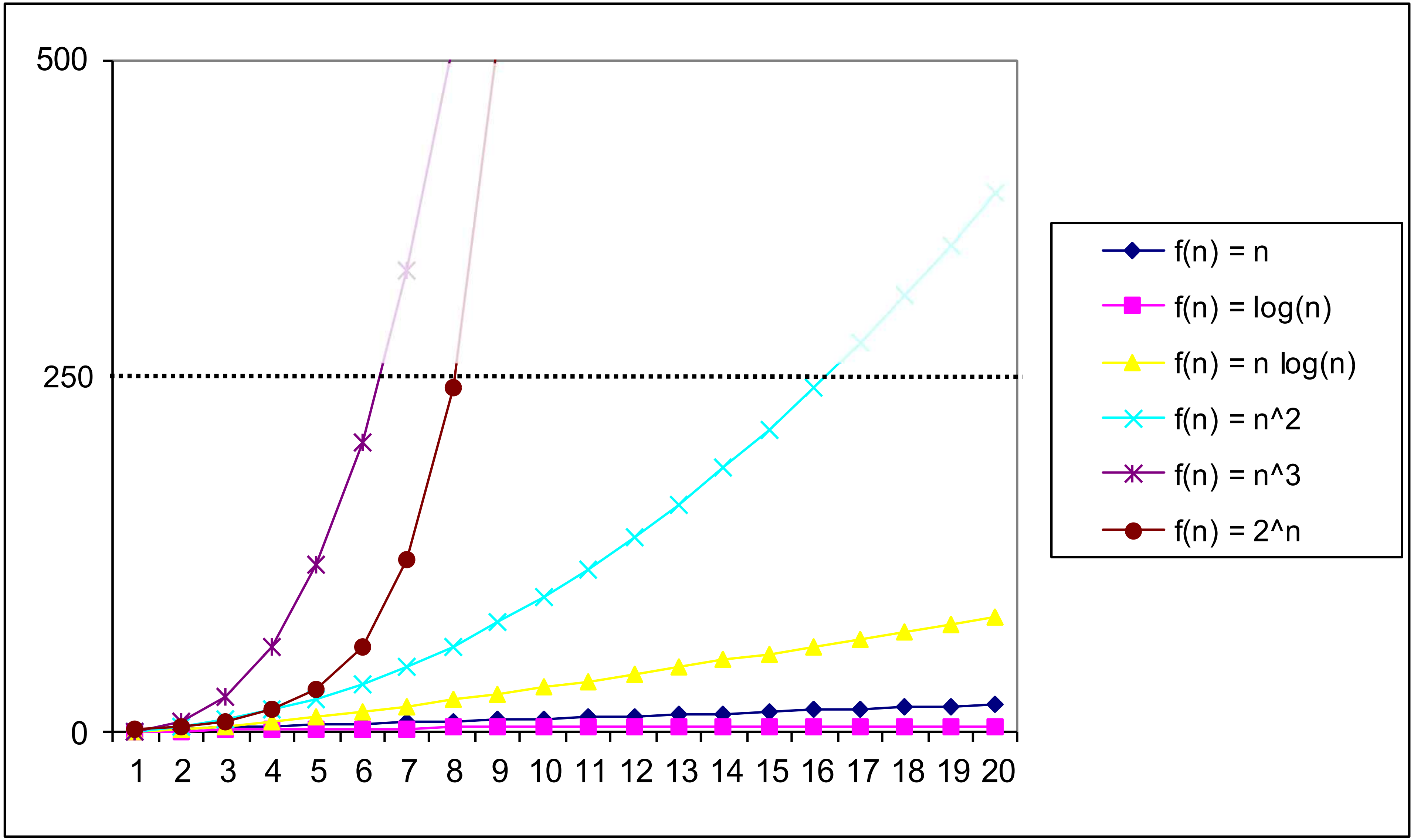
Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Practical Complexity



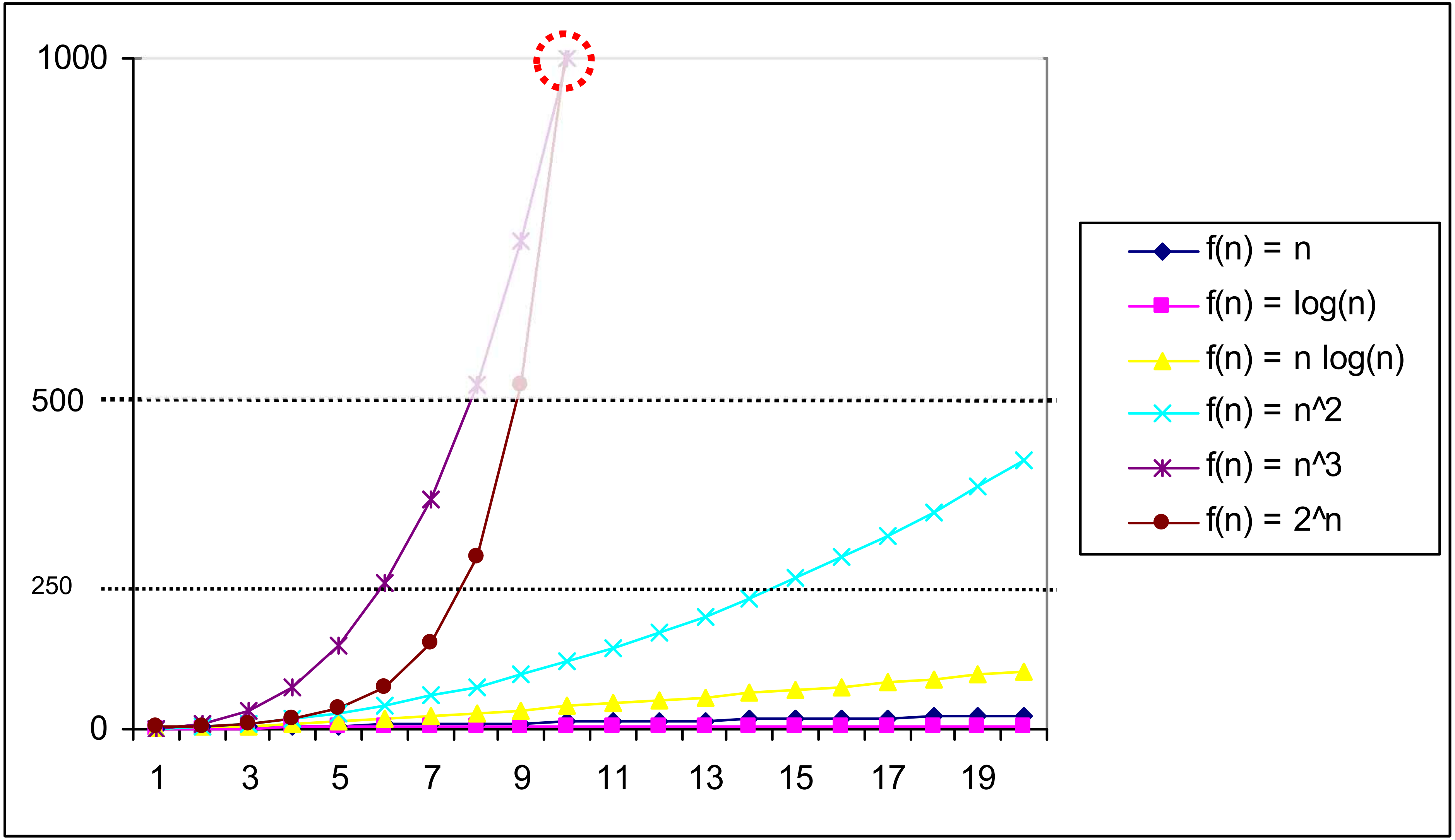
Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Practical Complexity



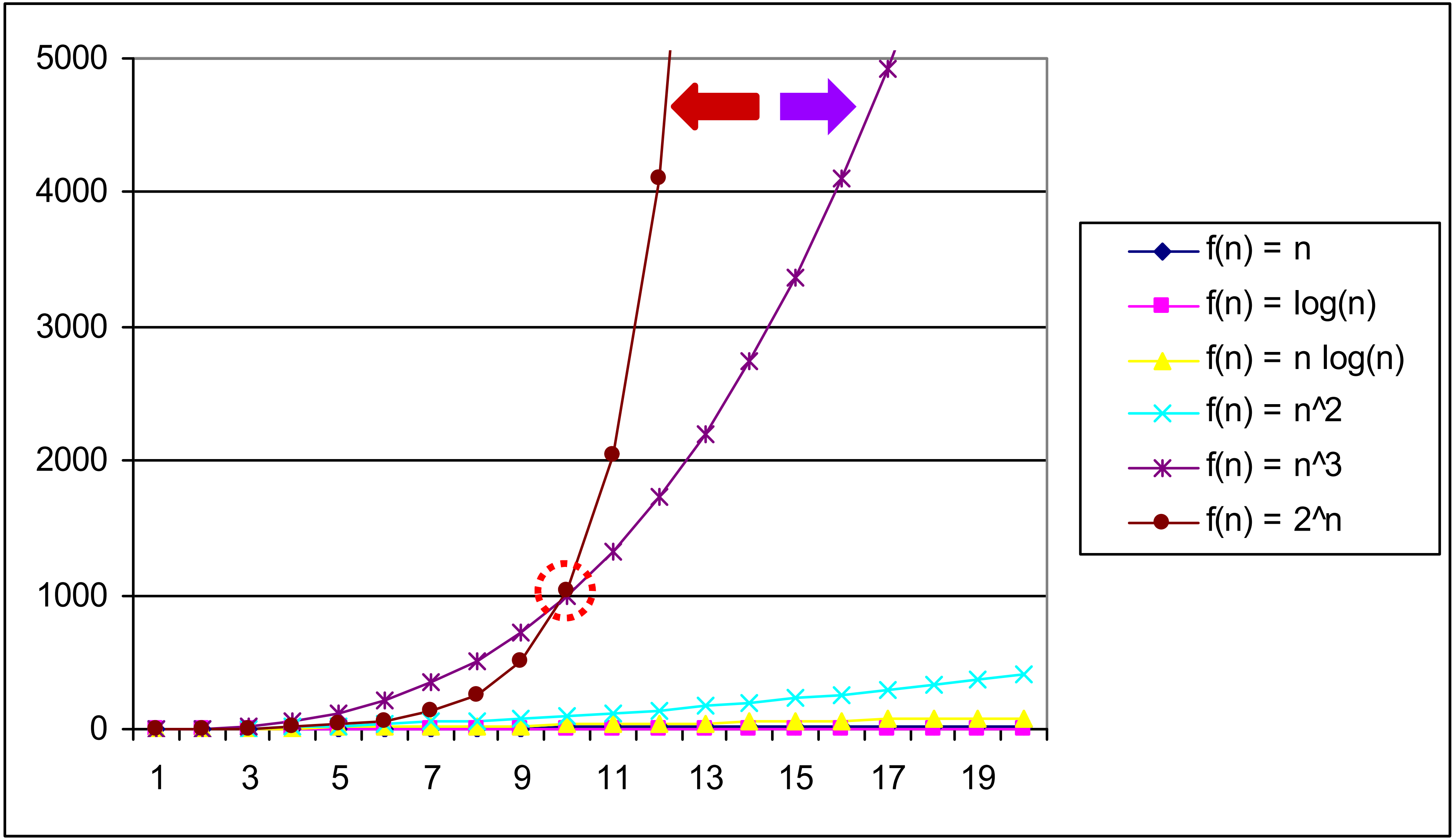
Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Practical Complexity



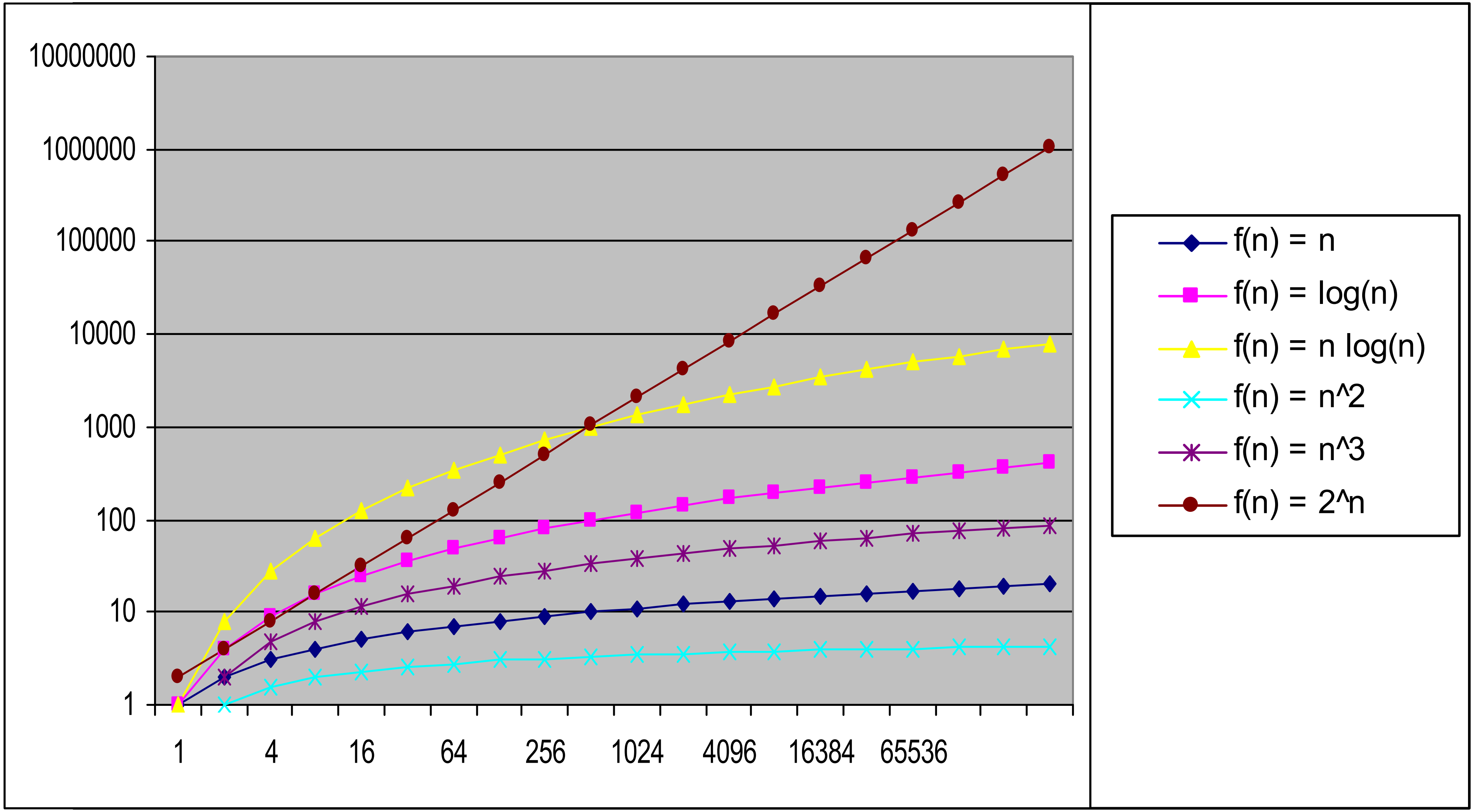
Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Practical Complexity



Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Practical Complexity



Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Ω-Omega notation: Lower Bound

- For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions
  - $f(n)$ is a member of $\Omega(g(n))$ $\exists$ positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n) \quad \forall\, n \ge n_0$
    - there existes some constant c s.t. always $c \cdot g(n) \le f(n)$ for large enough n.

- We use Ω-notation to give an asymptotic lower bound on a function, to within a constant factor.

Θ-notation: Asymptotic Tight Bound

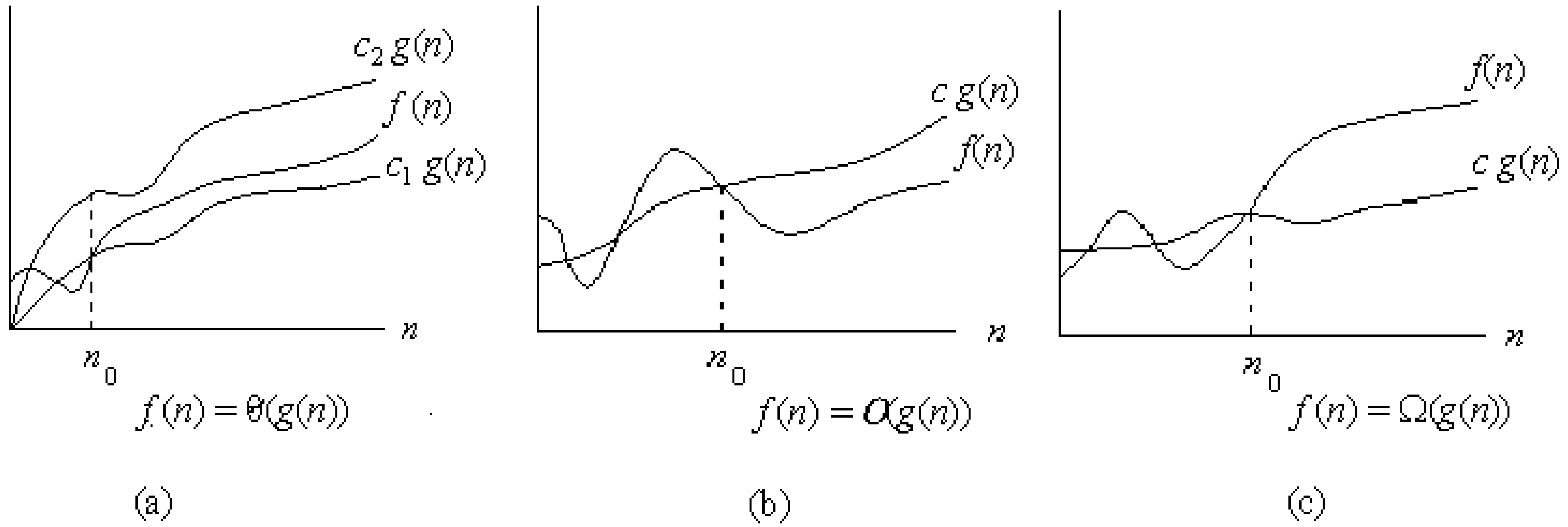
- A function $f(n)$ is $\Theta(\, g(n)\,)$ if $\exists$ positive constants c_1, c_2, and n_0 such that

$$c_1\, g(n) \le f(n) \le c_2\, g(n) \quad \forall\, n \ge n_0$$

- Theorem
  - $f(n)$ is $\Theta(\, g(n)\,)$ iff $f(n)$ is both $O(g(n))$ and $\Omega(g(n))$
  - Proof: someday

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Asymptotic notations Comparison



Graphic examples of Θ, O, *and* Ω.

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Example 1

Show that $f(n) = \frac{1}{2}n^2 - 3n = \Theta(n^2)$

We must find c_1 *and* c_2 *such that*

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

Dividing bothsides by n^2 *yields*

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

For $n_0 \geq 7$, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Example 2

$$f(n) = 3n^2 - 2n + 5 = \Theta(n^2)$$

because

$$3n^2 - 2n + 5 = \Omega(n^2)$$

$$3n^2 - 2n + 5 = O(n^2)$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Example 3

$3n^2 - 100n + 6 = O(n^2)$ since for $c = 3$, $3n^2 > 3n^2 - 100n + 6$

$3n^2 - 100n + 6 = O(n^3)$ since for $c = 1$, $n^3 > 3n^2 - 100n + 6$ when $n > 3$

$3n^2 - 100n + 6 \neq O(n)$ since for any c, $cn < 3n^2$ when $n > c$

$3n^2 - 100n + 6 = \Omega(n^2)$ since for $c = 2$, $2n^2 < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 \neq \Omega(n^3)$ since for $c = 3$, $3n^2 - 100n + 6 < n^3$ when $n > 3$

$3n^2 - 100n + 6 = \Omega(n)$ since for any c, $cn < 3n^2 - 100n + 6$ when $n > 100$

$3n^2 - 100n + 6 = \Theta(n^2)$ since both O and Ω apply.

$3n^2 - 100n + 6 \neq \Theta(n^3)$ since only O applies.

$3n^2 - 100n + 6 \neq \Theta(n)$ since only Ω applies.

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Other Asymptotic Notations

- A function $f(n)$ is $o(g(n))$ if $\exists$ positive constants c and n_0 such that

  $$f(n) < c\, g(n) \quad \forall\, n \geq n_0$$

- A function $f(n)$ is $\omega(g(n))$ if $\exists$ positive constants c and n_0 such that

  $$c\, g(n) < f(n) \quad \forall\, n \geq n_0$$

- Intuitively,
  - o() is like $<$
  - ω() is like $>$
  - Θ() is like $=$
  - O() is like $\leq$
  - Ω() is like $\geq$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Standard notations and common functions

- Factorials

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

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

For $n \geq 0$, the Stirling approximation:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

- Floors and ceilings

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Logarithms

For all real a>0, b>0, c>0, and n

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

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

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

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

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

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

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

$$\lg n = \log_2 n$$

$$\ln n = \log_e n$$

$$\log^k n = (\log n)^k$$

$$\lg \lg n = \lg(\lg n)$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Analyses of INSERTION, SELECTION, BUBBLE sort algorithms

ASYMPTOTIC ANALYSIS OF SIMPLE SORT ALGORITHMS

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Insertion Sort Analysis

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 $n \to \infty$

"Asymptotic Analysis"

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

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 (n), since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Insertion Sort Pseudocode

A pseudocode for insertion sort (INSERTION SORT).

```
INSERTION-SORT(A)
for j ← 2 to length [A]
      do key ← A[ j]
      ∇ Insert A[j] into the sorted sequence A[1,..., j-1].
      i ← j – 1
      while i > 0 and A[i] > key
            do A[i+1] ← A[i]
                 i ← i – 1
     A[i +1] ← key
```

How many times will this loop execute?
→ Depends on the condition

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Analysis of INSERTION-SORT

INSERTION-SORT(A)	cost	times
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
∇ Insert $A[j]$ into the sorted sequence $A[1 \cdot\cdot j-1]$	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ *and* $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Analysis of INSERTION-SORT

- The total running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) \\ &+ c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) \\ &+ c_8(n-1) \end{aligned}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

INSERTION-SORT: Best Case

- The best case: The array is already sorted.
  - Inner loop body never executed
  - $t_j = 1$ for $j=2,3,\ldots,n$
  - $T(n)$ is a linear function: $an + b$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + \\
&\quad c_5 \textstyle\sum_{j=2}^{n} 1 + c_6 \sum_{j=2}^{n}(1-1) + c_7 \sum_{j=2}^{n}(1-1) + \\
&\quad c_8(n-1) \\
&= c_1 n + c_2(n-1) + c_4(n-1) + \\
&\quad c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)(n-1) + c_1
\end{aligned}
$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

INSERTION-SORT: Worst Case

- The worst case: The array is in reversed order
  - Inner loop body executed for all previous elements
  - $t_j = j$ for $j=2,3, \ldots, n$
  - $T(n)$ is a quadratic function: $an^2 + bn + c$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + \\
&\quad c_5 \textstyle\sum_{j=2}^{n} j + c_6 \sum_{j=2}^{n} (j-1) + c_7 \sum_{j=2}^{n} (j-1) + \\
&\quad c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_8)(n-1) + c_1 + \\
&\quad c_5 \left\{ \frac{n(n+1)}{2} - 1 \right\} + (c_6 + c_7) \left\{ \frac{n(n-1)}{2} - 1 \right\} \\
&= \frac{c_5 + c_6 + c_7}{2} n^2 + (\ldots)n + \ldots
\end{aligned}
$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

INSERTION-SORT: Average Case

- The average case: The array is in random order
  - Inner loop body executed half the chance
  - $t_j = j/2$ for $j=2,3, \ldots, n$
  - $T(n)$ is also an quadratic function: $an^2 + bn + c$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + \\
&\quad c_5 \textstyle\sum_{j=2}^{n} \frac{j}{2} + c_6 \sum_{j=2}^{n} (\frac{j}{2} - 1) + c_7 \sum_{j=2}^{n} (\frac{j}{2} - 1) + \\
&\quad c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_8)(n-1) + c_1 + \\
&\quad c_5 \left\{ \frac{n(n+1)}{4} \ldots \right\} + (c_6 + c_7) \left\{ \frac{n(n-1)}{4} \ldots \right\} \\
&= \frac{c_5 + c_6 + c_7}{4} n^2 + (\ldots) n + \ldots
\end{aligned}
$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Insertion Sort Is O(n^2)

- Proof
  - Suppose runtime is $T(n) = an^2 + bn + c$
    - Here, we only consider positive a, b, and c

$$T(n) = an^2+bn+c \le (a+b+c)n^2 + (a+b+c)n + (a+b+c)$$
$$\le 3(a+b+c)n^2 \text{ for } n \ge 1$$

    - Let $c' = 3(a+b+c)$ and let $n_0 = 1$, then

$$T(n) \le c'n^2 \text{ for } n \ge n_0$$

- Question
  - Is InsertionSort O(n^3)? *YES*
  - Is InsertionSort O(n)? *NO*

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Selection Sort

Alg.: SELECTION-SORT(*A*)

```
n ← length[A]
for j ← 1 to n - 1
    do smallest ← j
       for i ← j + 1 to n
           do if A[i] < A[smallest]
                 then smallest ← i
       exchange A[j] ↔ A[smallest]
```

8	4	6	9	2	3	1

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Analysis of Selection Sort

Alg.: SELECTION-SORT(*A*)	*cost*	*times*
n ← length[A]	c_1	1
for j ← 1 **to** n - 1	c_2	n
do smallest ← j	c_3	$n-1$
for i ← j + 1 **to** n	c_4	$\sum_{j=1}^{n-1}(n-j+1)$
do if A[i] < A[smallest]	c_5	$\sum_{j=1}^{n-1}(n-j)$
then smallest ← i	c_6	$\sum_{j=1}^{n-1}(n-j)$
exchange A[j] ↔ A[smallest]	c_7	$n-1$

≈$n^2/2$ comparisons

≈n exchanges

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4 \sum_{j=1}^{n-1}(n-j+1) + c_5 \sum_{j=1}^{n-1}(n-j) + c_6 \sum_{j=2}^{n-1}(n-j) + c_7(n-1) \propto n^2 = \Theta(n^2)$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Bubble-Sort Running Time

Alg.: BUBBLESORT(A)

for $i \leftarrow 1$ **to** *length[A]* c_1

do for $j \leftarrow$ *length[A]* **downto** $i + 1$ c_2

Comparisons: $\approx n^2/2$ **do if** $A[j] < A[j-1]$ c_3

Exchanges: $\approx n^2/2$ **then** exchange $A[j] \leftrightarrow A[j-1]$ c_4

$$\begin{aligned} T(n) &= c_1(n+1) + c_2 \sum_{j=1}^{n}(n-j+1) \\ &\quad + c_3 \sum_{j=1}^{n}(n-j) + c_4 \sum_{j=1}^{n}(n-j) \\ &\approx c_1 n + (c_2 + c_3 + c_4) \sum_{j=1}^{n}(n-j) \\ &= c_1 n + (c_2 + c_3 + c_4) \cdot \frac{n(n-1)}{2} \\ &\propto n^2 = \Theta(n^2) \end{aligned}$$

MASTER THEOREM FOR ASYMPTOTIC ANALYSIS

Review: Asymptotic Notation

- Upper Bound Notation:
  - $f(n)$ is $O(g(n))$ if $\exists$ positive constants c and n_0 such that $f(n) \le c \cdot g(n)$ for all $n \ge n_0$
- Asymptotic lower bound:
  - $f(n)$ is $\Omega(g(n))$ if $\exists$ positive constants c and n_0 such that $0 \le c{\cdot}g(n) \le f(n)$ $\forall\, n \ge n_0$
- Asymptotic tight bound:
  - $f(n)$ is $\Theta(g(n))$ if $\exists$ positive constants c_1, c_2, and n_0 such that $c_1\, g(n) \le f(n) \le c_2\, g(n)$ $\forall\, n \ge n_0$
  - $f(n) = \Theta(g(n))$ if and only if [$f(n) = O(g(n))$ AND $f(n) = \Omega(g(n))$]

Review: Asymptotic Notations

- $f(n)$ is $o(g(n))$ if $\exists$ positive constants c and n_0 s. t.
  $f(n) < c\, g(n) \quad \forall\ n \geq n_0$
- $f(n)$ is $\omega(g(n))$ if $\exists$ positive constants c and n_0 s. t.
  $c\, g(n) < f(n) \quad \forall\ n \geq n_0$
- Big O fact:
  - A polynomial of degree k is $O(n^k)$
- Intuitively,

- o() is like <
- ω() is like >
- Θ() is like =
- O() is like ≤
- Ω() is like ≥

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Recurrences

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

is a *recurrence*.

- Recurrence: an equation that describes a function in terms of its value on smaller functions

More Examples of Recurrences

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + c & n > 0 \end{cases}$$

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

General form

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Solving Recurrences

- Mathematical induction
- Substitution method
- Iteration method
- Master method

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Mathematical Induction

- Suppose
  - Base condition: $S(k)$ is true for fixed constant k
    - Often $k = 0$ or 1
  - Inductive hypothesis: $S(n)$ is true
  - Then $S(n+1)$ is true for all $n \geq k$
- Then $S(n)$ is true for all $n \geq k$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Proof By Induction

- Claim: $S(n)$ is true for all $n \geq k$
- Basis:
  - Show formula is true when $n = k$
- Inductive hypothesis:
  - Assume formula is true for an arbitrary choice of n
- Step:
  - Show that formula is then true for $n+1$ (under the inductive hypothesis)

Induction: Gaussian Closed Form

- Prove $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Basis:
    - If $n = 0$, then $0 = 0(0+1) / 2$
  - Inductive hypothesis:
    - Assume $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Step (show true for n+1):

    $1 + 2 + \ldots + n + (n+1) = (1 + 2 + \ldots + n) + (n+1)$

    $= n(n+1)/2 + n+1 = [n(n+1) + 2(n+1)]/2$

    $= (n+1)(n+2)/2 = \underline{(n+1)}(\underline{(n+1)} + 1) / 2$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Induction: Geometric Closed Form

- Prove $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$

    $a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis:
    - Assume $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show true for n+1):

    $a^0 + a^1 + \ldots + a^{n+1} = (a^0 + a^1 + \ldots + a^n) + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + (a^{n+1})\cdot\{(a - 1)/(a - 1)\}$

    $= (a^{n+1} - 1 + a^{n+2} - a^{n+1})/(a - 1) = (a^{(n+1)+1} - 1)/(a - 1)$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Solving Recurrences: Substitution

- The substitution method
  - A.k.a. (also known as) the “making a good guess method”
  - *Guess the form of the answer*, then use *induction* to find the constants and show that solution works

- Examples:
  - $T(n) = 2T(n/2) + n$ → $\Theta(n \log_2 n)$
  - $T(n) = 2T(n/2 + 17) + n$ → $\Theta(n \log_2 n)$
  - $T(n) = 2T(n/2) + \Theta(n)$ → $\Theta(n \log_2 n)$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + c & n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + c \\ &= T(n-2) + c + c \\ &= T(n-2) + 2c \\ &= T(n-3) + 2c + c \\ &= T(n-3) + 3c \\ & \ldots \\ &= T(n-k) + kc \end{aligned}$$

- So far for $n \geq k$ we have $T(n) = T(n-k) + kc$
- What if $k = n$? $T(n) = T(0) + nc = nc$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + n & n > 0 \end{cases}$$

$$\begin{aligned} & T(n) \\ & = T(n-1) + n \\ & = T(n-2) + (n-1) + n \\ & = T(n-3) + (n-2) + (n-1) + n \\ & \dots \\ & = T(n-k) + (n-(k-1)) + \dots + n \\ & = T(n-k) + \textstyle\sum_{i=n-k+1}^{n} i \end{aligned}$$

- So far for $n \geq k$ we have

$$T(n) = T(n-k) + \sum_{i=n-k+1}^{n} i$$

- What if $k = n$?

$$T(n) = T(0) + \sum_{i=1}^{n} i$$

$$= 0 + \frac{n(n+1)}{2}$$

- Thus in general

$$T(n) = \frac{n(n+1)}{2}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= 2T\left(\tfrac{n}{2}\right) + c \\
&= 2\left(2T\left(\tfrac{n}{2^2}\right) + c\right) + c \\
&= 2^2T\left(\tfrac{n}{2^2}\right) + 2c + c \\
&= 2^2\left(2T\left(\tfrac{n}{2^3}\right) + c\right) + 3c \\
&= 2^3T\left(\tfrac{n}{2^3}\right) + 4c + 3c \\
&= 2^3T\left(\tfrac{n}{2^3}\right) + 7c \\
&= 2^4T\left(\tfrac{n}{2^4}\right) + 15c \\
&= \dots \\
&= 2^kT\left(\tfrac{n}{2^k}\right) + \left(2^k - 1\right)c
\end{aligned}$$

- So far for $n > 2k$ we have $T(n) = 2^kT\left(\frac{n}{2^k}\right) + (2^k - 1)c$
- What if $k = \log_2 n$?

$$\begin{aligned}
T(n) &= 2^{\log_2 n}T\left(\tfrac{n}{2^{\log_2 n}}\right) + \left(2^{\log_2 n} - 1\right)c \\
&= nT\left(\tfrac{n}{n}\right) + (n-1)c \\
&= nT(1) + (n-1)c \\
&= nc + (n-1)c = c(2n-1)
\end{aligned}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T\left(\tfrac{n}{2}\right) + cn \\ &= 2\left(2T\left(\tfrac{n}{2^2}\right) + \tfrac{cn}{2}\right) + cn \\ &= 2^2 T\left(\tfrac{n}{2^2}\right) + cn + cn \\ &= 2^2\left(2T\left(\tfrac{n}{2^3}\right) + \tfrac{cn}{2^2}\right) + 2cn \\ &= 2^3 T\left(\tfrac{n}{2^3}\right) + cn + 2cn \\ &= 2^3 T\left(\tfrac{n}{2^3}\right) + 3cn \\ &= 2^4 T\left(\tfrac{n}{2^4}\right) + 4cn \\ &= \ldots \\ &= 2^k T\left(\tfrac{n}{2^k}\right) + kcn \end{aligned}$$

- So far for $n > 2k$ we have
  $$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kcn$$
- What if $k = \log_2 n$?

$$\begin{aligned} T(n) &= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + cn\log_2 n \\ &= nT\left(\tfrac{n}{n}\right) + cn\log_2 n \\ &= nT(1) + cn\log_2 n \\ &= nc + cn\log_2 n = cn(\log_2 n + 1) \end{aligned}$$

Solving Recurrences: Iteration

- Another option is what the book calls the "iteration method"
  - Expand the recurrence
  - Work some algebra to express as a summation
  - Evaluate the summation

- Derive a general complexity equation from:

$$\boxed{T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= aT\left(\tfrac{n}{b}\right) + cn \\
&= a\left(aT\left(\tfrac{n}{b^2}\right) + \tfrac{cn}{b}\right) + cn \\
&= a^2 T\left(\tfrac{n}{b^2}\right) + cn\tfrac{a}{b} + cn \\
&= a^2 T\left(\tfrac{n}{b^2}\right) + cn\left(\tfrac{a}{b} + 1\right) \\
&= a^2\left(aT\left(\tfrac{n}{b^3}\right) + \tfrac{cn}{b^2}\right) + cn\left(\tfrac{a}{b} + 1\right) \\
&= a^3 T\left(\tfrac{n}{b^3}\right) + cn\left(\tfrac{a^2}{b^2} + \tfrac{a}{b} + 1\right) \\
&= \ldots \\
&= a^k T\left(\tfrac{n}{b^k}\right) + cn\sum_{i=0}^{k-1}\left(\tfrac{a}{b}\right)^i
\end{aligned}$$

For $k = \log_b n \quad (\Leftrightarrow n = b^k)$,

$$\begin{aligned}
T(n) &= a^k T\left(\tfrac{n}{n}\right) + cn\sum_{i=0}^{k-1}\left(\tfrac{a}{b}\right)^i \\
&= ca^k + cn\sum_{i=0}^{k-1}\left(\tfrac{a}{b}\right)^i \\
&= ca^k\tfrac{n}{b^k} + cn\sum_{i=0}^{k-1}\left(\tfrac{a}{b}\right)^i \\
&= cn\tfrac{a^k}{b^k} + cn\sum_{i=0}^{k-1}\left(\tfrac{a}{b}\right)^i \\
&= cn\sum_{i=0}^{k}\left(\tfrac{a}{b}\right)^i
\end{aligned}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = aT\left(\frac{n}{b}\right) + cn = cn\sum_{i=0}^{k}\left(\frac{a}{b}\right)^{i}, \qquad \forall n \geq 1, k = \log_b n$$

- What if $a = b$?

$$\begin{aligned} T(n) &= cn\,(k+1) \\ &= cn\,(\log_b n + 1) \\ &= \Theta(n \log n) \end{aligned}$$

- What if $a < b$?

$$\sum_{i=0}^{k} r^i = \frac{r^{k+1}-1}{r-1}$$

$$\begin{aligned} T(n) &= cn\sum_{i=0}^{k}\left(\frac{a}{b}\right)^{i} \\ &= cn\frac{(a/b)^{k+1}-1}{(a/b)-1} = cn\frac{1-(a/b)^{k+1}}{1-(a/b)} \\ &< cn\frac{1}{1-(a/b)} \\ &= cn\Theta(1) = \Theta(n) \end{aligned}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

$$T(n) = aT\left(\frac{n}{b}\right) + cn = cn\sum_{i=0}^{k}\left(\frac{a}{b}\right)^i, \qquad \forall n \geq 1, k = \log_b n$$

- What if $a > b$?

$$\text{For } k = \log_b n$$

$$\sum_{i=0}^{k}\left(\frac{a}{b}\right)^i = \frac{(a/b)^{k+1}-1}{(a/b)-1} = \Theta\left(\left(\frac{a}{b}\right)^k\right)$$

$$T(n) = cn \cdot \Theta\left(\frac{a^k}{b^k}\right) = cn \cdot \Theta\left(\frac{a^{\log_b n}}{b^{\log_b n}}\right) = cn \cdot \Theta\left(\frac{a^{\log_b n}}{n}\right) = \Theta\left(a^{\log_b n}\right)$$

$$a^{\log n} = n^{\log a} \quad (\because \log a^{\log n} = \log n \cdot \log a = \log n^{\log a})$$

$$\Rightarrow T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log a}\right)$$

$$T(n) = aT\left(\frac{n}{b}\right) + cn = cn\sum_{i=0}^{k}\left(\frac{a}{b}\right)^{i}, \qquad \forall n \geq 1, k = \log_b n$$

For $a > 0$, $b > 0$,

$$T(n) = \begin{cases} \Theta(n) & a < b \\ \Theta(n \log_b n) & a = b \\ \Theta\left(n^{\log_b a}\right) & a > b \end{cases}$$

The Master Theorem

- Given: a *divide and conquer* algorithm
  - An algorithm that divides the problem of size n into a subproblems, each of size n/b
  - Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function $f(n)$
- Then, **the Master Theorem** gives us a cookbook for the algorithm's running time.

The Master Theorem

If $T(n) = aT(n/b) + f(n)$,

for $\underline{\varepsilon > 0}$, $\underline{c < 1}$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right) & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a} \cdot \log n\right) & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(f(n)\right) & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ AND} \\ & af(n/b) < cf(n) \text{ for large } n \end{cases}$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Using The Master Theorem

$$
\begin{aligned}
T(n) = \quad & 9T(n/3) + n \\
& a = 9,\ b = 3,\ f(n) = n \\
& n^{\log_b a} = n^{\log_3 9} = n^2 \\
\text{Since} \quad & f(n) = O(n^2) = O(n^{(\log_3 9)-\varepsilon}) \text{ where } \varepsilon = 1 \\
\text{Apply} \quad & T(n) = \Theta\left(n^{\log_b a}\right) \text{ when } f(n) = O\left(n^{(\log_b a)-\varepsilon}\right)
\end{aligned}
$$

$$
T(n) = \quad \Theta(n^2)
$$

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I

Asymptotic performance analysis

END OF LECTURE 4

Slide credit: J. Lillis, UIC's CS 201 Data Structures and Discrete Mathematics I