DATA STRUCTURE

Lecture 01: Course Introduction, administration, etc.

Spring 2022

Hsiao-chi Li 李曉祺

ADMINISTRATIVE STUFF

- Who’s here? Where? And When?
  - 電機一乙 共同413(e) at 16:10 – 19:00
  - 電機一丙 共同313(e) at 13:10 – 16:00
- Contact Information
  - 1. Email: hcli@mail.ntut.edu.tw
  - 2. Office: 綜科 518
  - 3. Teaching Assistant – to be determined
- Office Hour: Tue. & Wed. 10:10 – 12:00

- Cell Phones should be silenced or turned off
- Texting and web surfing are never appropriate

GRADING, ETC.

- Grading:
  - Homework 15%
  - Quiz 10%
  - Midterm 30%
  - Final 35%
  - Attendance 10%

- Class Policy:
  - Cooperative learning is wonderful and encouraged!
  - Cooperative testing is terrible and not encouraged!

GRADING, ETC. (CONT'D)

- Grading Policy
  - Late Submission: You can turn in your assignments late for no more than three days. There will be no penalty for your late submission within these three days. Once you used up the 5-day extension, your submission will be counted for no credits.
  - Example: If there are four assignments in this semester, and for these four you turn in your works
    - HW1 – 2 day later
    - HW2 – 3 day later (2+3=5 → no more extension for later assignments)
    - HW3 – on time
    - HW4 – 1 day later (not counted)

    No credits will be counted for the last assignment.

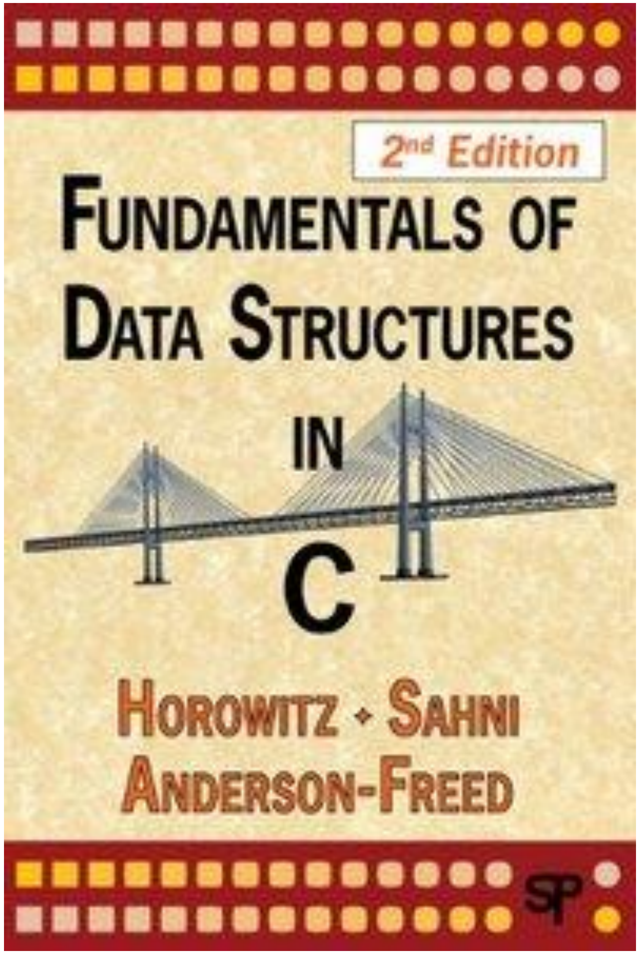
GRADING, ETC. (CONT'D)

- Policy on Copying
  - Copying is not acceptable. Do your own work.

- Attendance
  - Absence: 2 points of your final grade will be taking off.

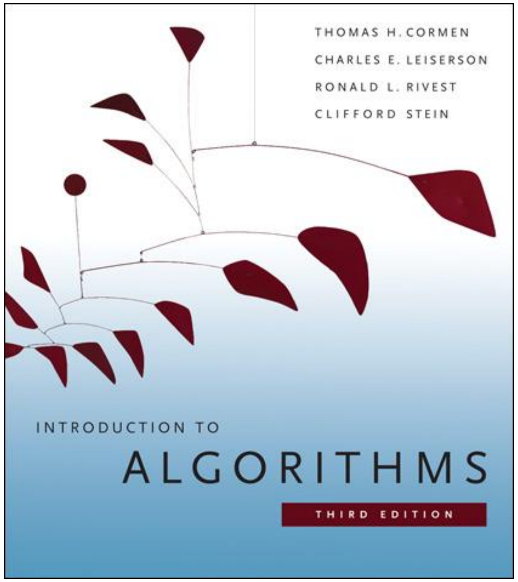
TEXT

- Horowitz, *Fundamentals of Data Structures in C*, 2nd edition, Silicon Press



REFERENCE

- Horowitz, Ellis, S. Sahni, and S. Rajasekaran, *Computer Algorithms / C++*. Summit, NJ: Silicon Press, 2007.
- M.T, Goodrich and R. Tamassia, *Algorithm Design*, John Wiley & Sons.
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd Edition, The MIT Press, 2009.



COURSE PLAN (1)

電機一乙

週次 Week	日期 Date	主題 Topic
1	02/21	Introduction & Overview
2	02/28	和平紀念日
3	03/07	Algorithms: Analysis, complexity, and the lower bound problem
4	03/14	Algorithms: Analysis, complexity, and the lower bound problem
5	03/21	Stacks, Queues, Trees, Dictionaries
6	03/28	Stacks, Queues, Trees, Dictionaries
7	04/04	兒童節
8	04/11	Graphs
9	04/18	Midterm

COURSE PLAN (2)

電機一乙

週次 Week	日期 Date	主題 Topic
10	04/25	Heaps, Sets
11	05/02	Sorting
12	05/09	The Greedy Method
13	05/16	The Divide-and-Conquer Strategy
14	05/23	Tree Searching Strategies; Prune and Search
15	05/30	Dynamic Programming
16	06/06	Shortest Paths
17	06/13	The Theory of NP-Completeness
18	06/20	Final

- Rough plan (Subject to change!)

COURSE PLAN (1)

電機一丙

週次 Week	日期 Date	主題 Topic
1	02/23	Introduction & Overview
2	03/02	Algorithms: Analysis, complexity, and the lower bound problem
3	03/09	Algorithms: Analysis, complexity, and the lower bound problem
4	03/16	Stacks, Queues, Trees, Dictionaries
5	03/23	運動會停課 (遇雨上課，依體育室公告為主)
6	03/30	Stacks, Queues, Trees, Dictionaries
7	04/06	校慶補假
8	04/13	Graphs
9	04/20	Midterm Exam

COURSE PLAN (2)

電機一丙

週次 Week	日期 Date	主題 Topic
10	04/27	Heaps, Sets
11	05/04	Sorting
12	05/11	The Greedy Method
13	05/18	The Divide-and-Conquer Strategy
14	05/25	Tree Searching Strategies; Prune and Search
15	06/01	Dynamic Programming
16	06/08	Shortest Paths
17	06/15	The Theory of NP-Completeness
18	06/22	Final

- Rough plan (Subject to change!)

BASIC CONCEPT

- What Is Data Structure?
- What Is Algorithm?

- Overview
  - System Life Cycle
  - Data Abstraction and Encapsulation
  - Algorithm Specification
  - Performance Analysis and Measurement

SYSTEM LIFE CYCLE

- Requirements

- Analysis
  - Bottom-up
  - Top-down

- Design
  - Data objects: abstract data types
  - Operations: specification & design of algorithms

SYSTEM LIFE CYCLE (CONT'D)

- Refinement & Coding
  - Choose representations for data objects
  - Write algorithms for each operation on data objects
- Verification
  - Correctness proofs: selecting proved algorithms
  - Testing: correctness & efficiency
  - Error removal: well-documented

EVALUATIVE JUDGEMENTS ABOUT PROGRAMS

- Meet the Original Specification?
- Work Correctly?
- Document?
- Use Functions to Create Logical Units?
- Code readable?
- Use Storage Efficiently?
- Running Time Acceptable?

DATA ABSTRACTION AND ENCAPSULATION

- Data encapsulation or information hiding is the concealing of the implementation details of a data object from the outside world
- Data abstraction is the separation between the specification of a data object and its implementation
- A data type is a collection of objects and a set of operations that act on those objects

DATA ABSTRACTION

- Specification
  - Name of function
  - Type of arguments
  - Type of result
  - Description of what the function does

Predefined data types
*Struct student { char last_name
int student_id
char grade; }

Data type: object & operation
integer: +,-,,/,%,=,==

- Representation
  - Implementation details
  - e.g. char 1 byte, int 4 bytes

DATA ABSTRACTION

- A abstract data type (ADT) is a data type that
  - is organized in such a way that the specification of the objects, and
  - the specification of the operations on the objects is separated from the representation of the objects and the implementation of the operations
- ADT is implementation-independent.

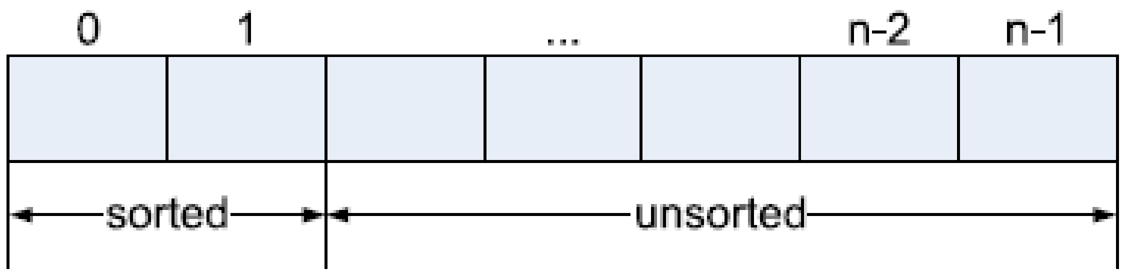
__Abstract data type NaturalNumber (p.9)__

```
ADT NaturalNumber is
  objects: an ordered subrange of the integers starting at
    zero and ending at the maximum integer (INT_MAX) on
    the computer
  functions:
    for all x, y ∈ Nat_Number; TRUE, FALSE ∈ Boolean
    and where +, -, <, and == are the usual integer
    operations.
    Zero (  ):NaturalNumber          ::=  0
    Is_Zero(x):Boolean               ::= if (x) return FALSE
                                         else return TRUE
    Add(x, y):NaturalNumber          ::= if ((x+y) <= INT_MAX)
                                           return x+y
                                         else return INT_MAX
    Equal(x,y):Boolean               ::= if (x== y) return TRUE
                                         else return FALSE
    Successor(x):NaturalNumber       ::= if (x == INT_MAX)
                                           return x
                                         else return x+1
    Subtract(x,y):NaturalNumber      ::= if (x<y) return 0
                                         else return x-y
end Natural_Number
```

ALGORITHM SPECIFICATION

- Definition
  - An algorithm is a finite set of instructions that accomplishes a particular task.
- Criteria
  - Input
  - Output
  - Definiteness: clear and unambiguous
  - Finiteness: terminate after a finite number of steps
  - Effectiveness: instruction is basic enough to be carried out

EXAMPLE 1: SELECTION SORT



- From those integers that are currently unsorted, find the smallest and place it next in the sorted list.

```
for ( i=0; i<n; i++) {
  Examine list[i] to list[n-1] and suppose
    that smallest integer is list[min]
  Interchange list[i] & list[min]
}
```

EXAMPLE 1: SELECTION SORT (CONT.)

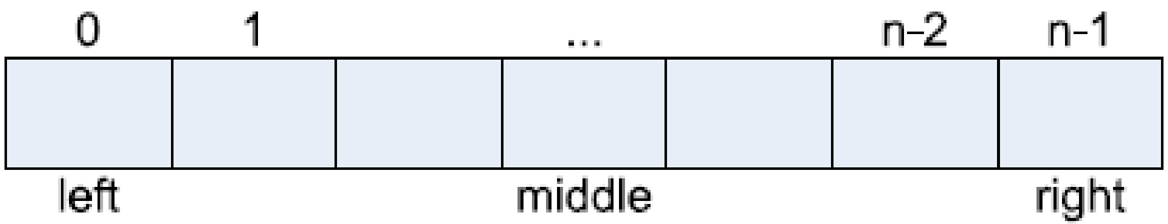
```
void sort(int list[ ], int n)
{
  for (i=0; i<n-1; i++)
  {
    int min = i;
    for (j=i+1; j<n; j++)
      if (list[j]<list[min])
        min=j;
      SWAP(list[i], list[min], temp);
  }
}
```

EXAMPLE OF SELECTION SORT

- Input: 20, 10, 15, 6, 17, 30

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
Original	20	10	15	6	17	30
Pass 0						
Pass 1						
Pass 2						
Pass 3						
Pass 4						

EXAMPLE 2: BINARY SEARCH




```
while (there are more integers to check)
{
  middle = (left + right) /2;
  if (searchnum < list[middle])
    right = middle -1;
  else if (searchnum == list[middle])
    return middle;
  else
    left = middle+1;
}
```

EXAMPLE 2: BINARY SEARCH (CONT.)

```
int compare(int x, int y)
/* return -1 for less than, 0 for equal */
int binsearch(int list[], int searchno, int left,
    int right)
{
  while (left <= right)  {
    middle = (left + right) / 2;
    switch ( COMPARE(list[middle], searchno) )  {
      case -1:
        left = middle +1;
        break;
      case 0:
        return middle;
      case 1:
        right = middle -1;
    }
  }
}
```

EXAMPLE OF BINARY SEARCH

- Input: 1,4,7,10,12,13,17,23,32

- Search for 10
- Search for 15

EXAMPLE 3: SELECTION PROBLEM

- Selection problem: select the k^{th} largest among N numbers
  - Approach 1
    - Read N numbers into an array
    - Sort the array in decreasing order
    - Return the element in position k

EXAMPLE 3: SELECTION PROBLEM (CONT.)

- Approach 2
  - Read *k* elements into an array
  - Sort them in decreasing order
  - For each remaining elements, read one by one
  - Ignored if it is smaller than the k^{th} element
  - Otherwise, place in correct place and bumping one out of array

- Which is better?
- Efficiency?

EXAMPLE OF SELECTION PROBLEM

- Input: 20, 9, 15, 6, 17, 30

- Find the third largest number

RECURSIVE ALGORITHMS

- Recursion is usually used to solve a problem in a *divided-and-conquer* manner
- Direct Recursion
  - Functions that call themselves
- Indirect Recursion
  - Functions that call other functions that invoke calling function again
- $C\binom{n}{m} = \frac{n!}{m!(n-m)!}$
  - $C\binom{n}{m} = C\binom{n-1}{m} + C\binom{n-1}{m-1}$
- Boundary condition for recursion

RECURSIVE SUMMATION

- sum(1,*n*) = sum(1,*n*-1)+*n*
- sum(1,1) = 1

```
int sum(int n)
{
  if (n==1)
    return (1);
  else
    return(sum(n-1)+n);
}
```

RECURSIVE FACTORIAL

- $n! = n\ (n-1)!$
- factorial(n) = $n\times$factorial(n-1)
- $0! = 1$

```
int fact(int n)
{
  if ( n== 0)
    return (1);
  else
  return (n*fact(n-1));
}
```

RECURSIVE MULTIPLICATION

- $a \times b = a \times (b - 1) + a$

```
int mult(int a, int b)
{
  if ( b==1)
    return (a);
  else
    return(mult(a,b-1)+a);
}
```

BINARY SEARCH

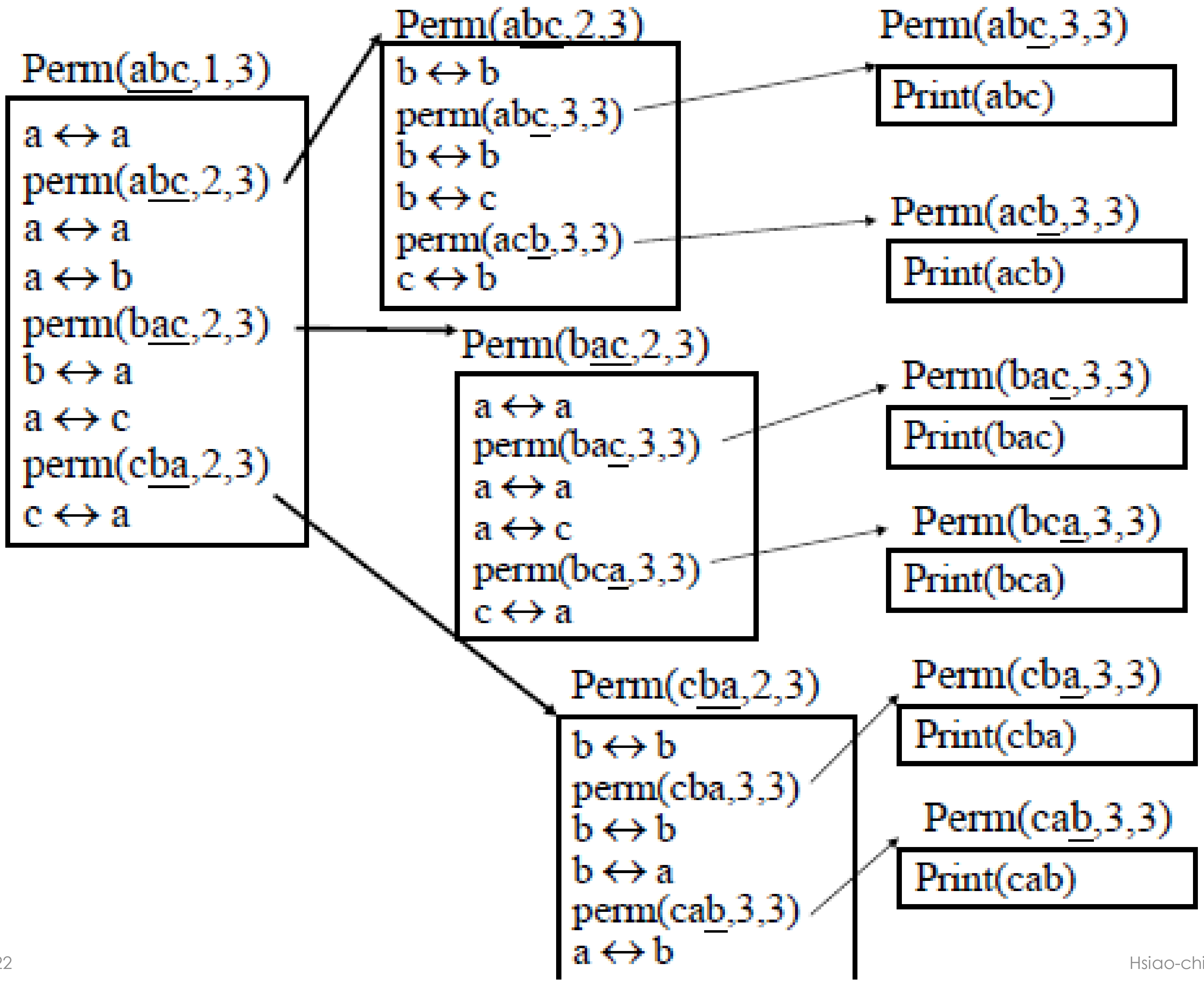
```
int compare(int x, int y)
/* return -1 for less than, 0 for equal */
int binsearch(int list[], int searchno, int left,
    int right)
{
  while (left <= right)  {
    middle = (left + right) / 2;
    switch ( COMPARE(list[middle], searchno) )  {
      case -1:
        left = middle +1;
        break;
      case 0:
        return middle;
      case 1:
        right = middle -1;
    }
  }
}
```

RECURSIVE BINARY SEARCH

```
int binsearch(int list[], int searchno, int left,
    int right)
{
  if (left <= right) {
    middle = (left + right)/2;
    switch (COMPARE(list[middle], searchno) ) {
      case -1:
        return binsearch(list, searchno, middle+1,
               right)
      case 0:
        return middle;
      case 1:
        return binsearch(list, searchno, left,
               middle-1);
    }
  }
  return -1;
}
```

RECURSIVE PERMUTATION

- Permutation of {a, b, c}
  - (a, b, c), (a, c, b)
  - (b, a, c), (b, c, a)
  - (c, a, b), (c, b, a)
- Recursion?
  - a+Perm({b,c})
  - b+Perm({a,c})
  - c+Perm({a,b})


Perm(abc,1,3)
a ↔ a
perm(abc,2,3)
a ↔ a
a ↔ b
perm(bac,2,3)
b ↔ a
a ↔ c
perm(cba,2,3)
c ↔ a
Perm(abc,2,3)
b ↔ b
perm(abc,3,3)
b ↔ b
b ↔ c
perm(acb,3,3)
c ↔ b
Perm(bac,2,3)
a ↔ a
perm(bac,3,3)
a ↔ a
a ↔ c
perm(bca,3,3)
c ↔ a
Perm(cba,2,3)
b ↔ b
perm(cba,3,3)
b ↔ b
b ↔ a
perm(cab,3,3)
a ↔ b
Perm(abc,3,3)
Print(abc)
Perm(acb,3,3)
Print(acb)
Perm(bac,3,3)
Print(bac)
Perm(bca,3,3)
Print(bca)
Perm(cba,3,3)
Print(cba)
Perm(cab,3,3)
Print(cab)

RECURSIVE PERMUTATION (CONT'D.)

```
void perm(char *list, int i, int n)
{
  if (i==n) {
    for (j=0; j<=n; j++)
      printf("%c", list[j]);
  }
  else {
    for (j=i; j<= n; j++) {
      SWAP(list[i], list[j], temp);
      perm(list, i+1, n);
      SWAP(list[i], list[j], temp);
    }
  }
}
```

PERFORMANCE EVALUATION

- Criteria
  - Is it correct?
  - Is it readable?
- Performance analysis
  - Machine Independent
- Performance measurement
  - Machine dependent

PERFORMANCE ANALYSIS

- Complexity theory
- Space Complexity
  - Amount of memory
- Time Complexity
  - Amount of computing time

SPACE COMPLEXITY

- $S(P) = c + S_p(I)$
  - c: fixed space (instruction, simple variables, constants)
  - $S_p(I)$: depends on characteristics of instance I
    - Characteristics: number, size, values of I/O associated with I
- If n is the only characteristic, $S_p(I) \Rightarrow S_p(n)$

SPACE COMPLEXITY (CONT'D.)

```
float abc(float a, float b, float c)
{
    return a+b+b*c+(a+b-c)/(a+b)+4.00;
}
```

$\boxed{S_{abc}(I)=0}$

SPACE COMPLEXITY (CONT'D.)

```
float rsum(float list[ ], int n)
{
  if (n)
    return rsum(list, n-1) + list[n-1];
  return 0;
 }
```

$S_{sum}(I)=S_{sum}(n)=6n$

Assumptions:

Space needed for one recursive call of the program

Type	Name	Number of bytes
Parameter: float Parameter: integer Return address: (used internally)	list[ ] n	2 2 2 (unless a far address)
Total		6

TIME COMPLEXITY

- $T(P) = c + T_p(I)$
  - c: compile time
  - $T_p(I)$: program execution time
    - Depends on characteristics of instance *I*

- Predict the growth in run times as the instance characteristics change

TIME COMPLEXITY (CONT'D.)

- Compile time (c)
  - Independent of instance characteristics

- Run (execution) time T_P
  - Real measurement
  - Analysis: counts of program steps

- Definition

  A program step is a syntactically or semantically meaningful program segment whose execution time is independent of the instance characteristics.

METHODS TO COMPUTE THE STEP COUNT

- Introduce variable count into programs
- Tabular method
- Determine the total number of steps contributed by each statement

  step per execution × frequency
- Add up the contribution of all statements

TIME COMPLEXITY (CONT'D.)

```
float sum(float list[ ], int n)
{
  float tempsum = 0;
  count++;         /* for assignment */
  int i;
  for (i=0; i<n; i++) {
    count++;       /* for the for loop */
    tempsum += list[i];
    count++;       /* for assignment */
  }
  count++;         /* last execution of for */
  return tempsum;
  count++;         /* for return */
}
```

2n+3 steps

TIME COMPLEXITY (CONT'D.)

```
float rsum(float list[ ], int n)
{
  count++;
  /* for if conditional */
  if (n<=0) {
    count++; // for return
    return 0
  }
  else {
    count++; // for return
    return rsum(list, n-1) + list[n-1];
  }
  count++;
  return list[0];
}
```

$$\boxed{\begin{aligned} &T(n) \\ &=2+T(n-1) \\ &=2+2+T(n-2) \\ &\ldots \\ &=2n+T(0) \\ &=2n+2 \end{aligned}}$$

TABULAR METHOD

Table 1.1: Step count table for Program 1.13 (p.40)

Statement	s/e	Frequency	Total steps
`float sum(float list[ ],`			
`int n)`			
`{`	0	1	0
`float tempsum = 0;`	1	1	1
`for(int i=0; i <n; i++)`	1	n+1	n+1
`tempsum += list[i];`	1	n	n
`return tempsum;`	1	1	1
`}`	0	1	0
Total			2n+3

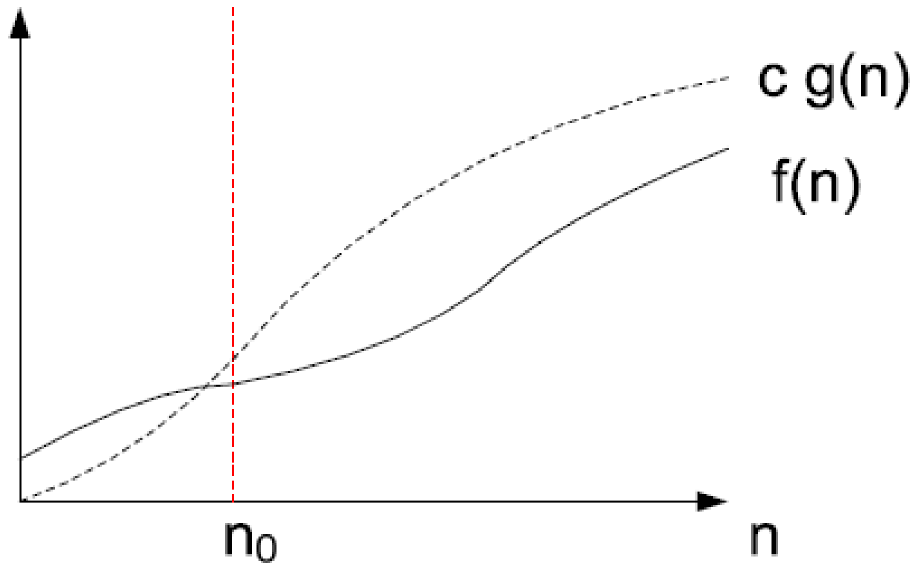
s/e: steps per execution

TIME COMPLEXITY (CONT'D.)

- Difficult to determine the exact step counts
- What a step stands for is inexact
  eg. x := y versus x := y + z + (x/y) + …
- Exact step count is not useful for comparison
- Step count doesn't tell how much time step takes
- Just consider the growth in run time

ASYMPTOTIC NOTATION – BIG "OH"

- $f(n) = O\big(g(n)\big)$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$



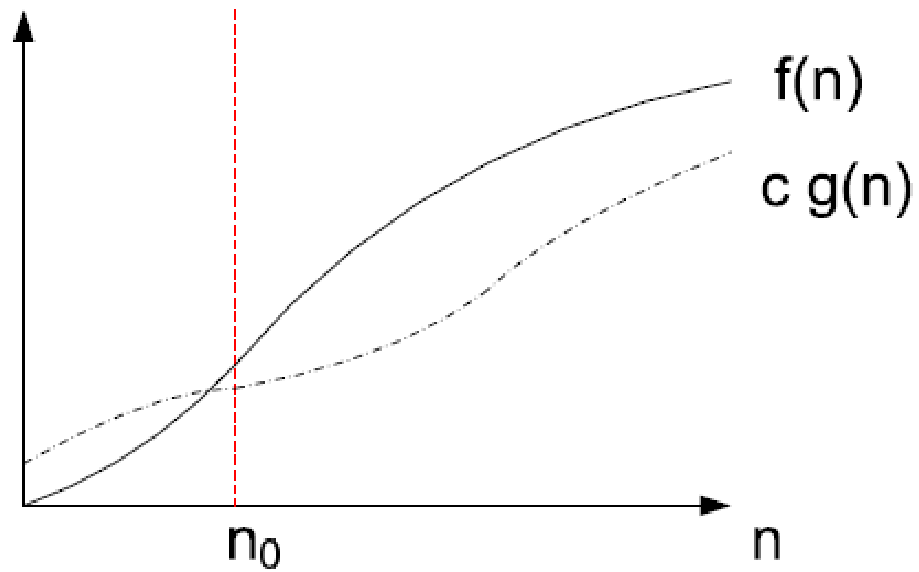
ASYMPTOTIC NOTATION – BIG "OH" (CONT'D.)

- $f(n) = O\big(g(n)\big)$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \leq c \cdot g(n)$, $\forall n \geq n_0$
  - eg.
    - $3n + 6 = O(n)$
    - $4n^2 + 2n - 6 = O\big(n^2\big)$
    - $f(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$
    $f(n) = O(n^m)$
- $g(n)$ should be a least upper bound.

ASYMPTOTIC NOTATION - OMEGA

- $f(n) = \Omega\big(g(n)\big)$ iff
  - $\exists$ a real constant $c > 0$ and an integer constant $n_0 \geq 1$, s.t. $f(n) \geq c \cdot g(n)$, $\forall n \geq n_0$

- $g(n)$ should be a most lower bound.



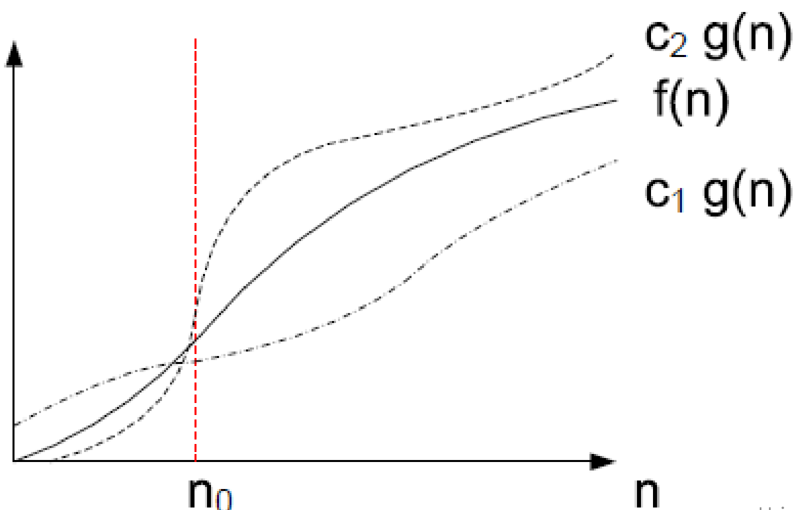
ASYMPTOTIC NOTATION – OMEGA (CONT'D)

- eg.
  - $3n+3 = \Omega(n)$
  - $3n^2+4n-8 = \Omega(n^2)$
  - $6*2^n+n^2 = \Omega(2^n)$

ASYMPTOTIC NOTATION - THETA

- $f(n) = \Theta(g(n))$ iff
  - $\exists$ real constants c_1 and $c_2 > 0$ and an integer constant $n_0 \geq 1$, s.t. $c_1 g(n) \leq f(n) \leq c_2 g(n)$, $\forall n \geq n_0$

- $g(n)$ should be both upper bound and lower bound. It is called precise bound.



ASYMPTOTIC NOTATION – THETA (CONT'D)

- eg.
  - f(n) = $3n^2+4n-8$
  - f(n) = log(n!)

- $f(n) = \Theta\big(g(n)\big) \Leftrightarrow f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$