

Welcome To ...

Data Structures & OO

王豐堅
教授 資工系

Office EC#518

Course Hours: Mon EF, Thur. B,
EC#015

Office Hours:

- 2:30-4:30pm Tues&Thur,
- Room: 工三館, EC#518

TA Hours:

- 6:00-9:00pm Tuesday (practice)
- 1:00-3:00pm, Thurs, 10:00-1200am, Fri, EC#510

Text Book and References

Text Book:

- Fundamentals of Data Structures, 2nd version, 2007, Silicon Express,
- by Howrowitz, Sahni, Mehta

References:

- Classical Data Structures in C++, 1994, Addison-Wesley
- By Budd, Timothy A.
- ..., etc.

2nd Edition

FUNDAMENTALS OF DATA STRUCTURES

IN

C++

HOROWITZ ♦ SAHNI ♦ MEHRA

SP

Clip Art Sources

www.barrysclipart.com

www.livinggraphics.com

www.rad.kumc.edu

www.graphicmaps.com

What The Course Is About

Data structures is concerned with the representation and manipulation of data.


All programs manipulate data.

So, all programs represent data in some way.

Data manipulation requires an algorithm.

What The Course Is About

- We shall study ways to represent data and algorithms to manipulate these representations.

- The study of data structures is
 fundamental to Computer Science & Engineering.

Course Outlines

Introduction to C++ and Algorithm

C++ and Arrays

Arrays (Strings)

Stacks and Queues

Linked Lists(Singly and doubly linked)

Linked Lists

Tree (Basic facts, binary trees)

Trees (search and heap)

Mid Term

Course Outlines (c.)

Graphs (basic facts, representations)

Graphs (shortest paths, spanning trees, topological sorting)

Internal Sorting (insertion, quick, and merge)

Internal Sorting (heap, radix, and the rest)

Hashing

Selected Advanced Topics

Final Exam

Prerequisites

C++

Asymptotic Complexity

- Big Oh, Theta, and Omega notations

Web Site

<e3>

Handouts, syllabus, text, source codes, exercise solutions, lectures, assignments, TAs, etc.



My office data.

Grades

15% for 3 home assignments

21% for 3 program assignments

25% for midterm test

39% for final test

Grades (Rough Cutoffs)

A $\geq 83\%$

B+ $\geq 75\%$

B $\geq 70\%$

C+ $\geq 65\%$

C $\geq 60\%$

... the rest

Sorting

Rearrange $a[0], a[1], \dots, a[n-1]$ into ascending order. When done, $a[0] \leq a[1] \leq \dots \leq a[n-1]$

8, 6, 9, 4, 3 \Rightarrow 3, 4, 6, 8, 9

Sort Methods

Insertion Sort

Bubble Sort

Selection Sort

Count Sort

Shaker Sort

Shell Sort

Heap Sort

Merge Sort

Quick Sort

Insert An Element

Given a sorted list/sequence, insert a new element

Given 3, 6, 9, 14

Insert 5

Result 3, 5, 6, 9, 14

Insert an Element

3, 6, 9, 14 insert 5

Compare new element (5) and last one (14)

Shift 14 right to get 3, 6, 9, , 14

Shift 9 right to get 3, 6, , 9, 14

Shift 6 right to get 3, , 6, 9, 14

Insert 5 to get 3, 5, 6, 9, 14

Insert An Element

```
// insert t into a[0:i-1]
```

```
int j;
```

```
for (j = i - 1; j >= 0 && t < a[j]; j--)
```

```
    a[j + 1] = a[j];
```

```
a[j + 1] = t;
```

Insertion Sort

Start with a sequence of size 1

Repeatedly insert remaining elements

Insertion Sort

Sort 7, 3, 5, 6, 1

Start with 7 and insert 3 \Rightarrow 3, 7

Insert 5 \Rightarrow 3, 5, 7

Insert 6 \Rightarrow 3, 5, 6, 7

Insert 1 \Rightarrow 1, 3, 5, 6, 7

Insertion Sort

```
for (int i = 1; i < a.length; i++)  
{// insert a[i] into a[0:i-1]  
    // code to insert comes here  
}
```

Insertion Sort

```
for (int i = 1; i < a.length; i++)  
{// insert a[i] into a[0:i-1]  
    int t = a[i];  
    int j;  
    for (j = i - 1; j >= 0 && t < a[j]; j--)  
        a[j + 1] = a[j];  
    a[j + 1] = t;  
}
```

---Complexity

Space/Memory

Time

- Count a particular operation
- Count number of steps
- Asymptotic complexity

Comparison Count

```
for (int i = 1; i < a.length; i++)  
{// insert a[i] into a[0:i-1]  
    int t = a[i];  
    int j;  
    for (j = i - 1; j >= 0 && t < a[j]; j--)  
        a[j + 1] = a[j];  
    a[j + 1] = t;  
}
```


Comparison Count

Pick an instance characteristic ... n , $n = a.length$ for insertion sort

Determine count as a function of this instance characteristic.

Comparison Count

```
for (j = i - 1; j >= 0 && t < a[j]; j--)  
    a[j + 1] = a[j];
```

How many comparisons are made?

Comparison Count

```
for (j = i - 1; j >= 0 && t < a[j]; j--)  
    a[j + 1] = a[j];
```

number of compares depends on
a[]s and t as well as on i

Comparison Count

- Worst-case count = maximum count
- Best-case count = minimum count
- Average count

Worst-Case Comparison Count

```
for (j = i - 1; j >= 0 && t < a[j]; j--)  
    a[j + 1] = a[j];
```

$a = [1, 2, 3, 4]$ and $t = 0 \Rightarrow 4$ compares

$a = [1, 2, 3, \dots, i]$ and $t = 0 \Rightarrow i$ compares

Worst-Case Comparison Count

```
for (int i = 1; i < n; i++)  
    for (j = i - 1; j >= 0 && t < a[j]; j--)  
        a[j + 1] = a[j];
```

$$\begin{aligned}\text{total compares} &= 1 + 2 + 3 + \dots + (n-1) \\ &= (n-1)n/2\end{aligned}$$

Step Count

A step is an amount of computing that does not depend on the instance characteristic n

10 adds, 100 subtracts, 1000 multiplies can all be counted as a single step

n adds cannot be counted as 1 step

Step Count

s/e

for (int i = 1; i < a.length; i++)	1
{// insert a[i] into a[0:i-1]	0
int t = a[i];	1
int j;	0
for (j = i - 1; j >= 0 && t < a[j]; j--)	1
a[j + 1] = a[j];	1
a[j + 1] = t;	1
}	0

Step Count

s/e isn't always 0 or 1

```
x = sum(a, n);
```

where n is the instance characteristic and
sum adds $a[0:n-1]$ has a s/e count of n

Step Count

s/e steps

```
for (int i = 1; i < a.length; i++)
```

1

```
{// insert a[i] into a[0:i-1]
```

0

```
    int t = a[i];
```

1

```
    int j;
```

0

```
    for (j = i - 1; j >= 0 && t < a[j]; j--)
```

1

i+ 1

```
        a[j + 1] = a[j];
```

1

i

```
    a[j + 1] = t;
```

1

```
}
```

0

Step Count

```
for (int i = 1; i < a.length; i++)  
{ 2i + 3}
```

step count for

```
    for (int i = 1; i < a.length; i++)
```

is n

step count for body of for loop is

$$2(1+2+3+\dots+n-1) + 3(n-1)$$

$$= (n-1)n + 3(n-1)$$

$$= (n-1)(n+3)$$

Asymptotic Complexity of Insertion Sort

$O(n^2)$

What does this mean?

Complexity of Insertion Sort

Time or number of operations does not exceed $c \cdot n^2$ on any input of size n (n suitably large).

Actually, the worst-case time is $\Theta(n^2)$ and the best-case is $\Theta(n)$

So, the worst-case time is expected to quadruple each time n is doubled

Complexity of Insertion Sort

Is **$O(n^2)$** too much time?

Is the algorithm practical?

Practical Complexities

10^9 instructions/second

<i>n</i>	<i>n</i>	<i>$n \log n$</i>	<i>n^2</i>	<i>n^3</i>
<i>1000</i>	1mic	10mic	1milli	1sec
<i>10000</i>	10mic	130mic	100milli	17min
<i>10^6</i>	1milli	20milli	17min	32years

Impractical Complexities

10^9 instructions/second

n	n^4	n^{10}	2^n
1000	17min	3.2×10^{13} years	3.2×10^{283} years
10000	116 days	???	???
10^6	3×10^7 years	??????	??????

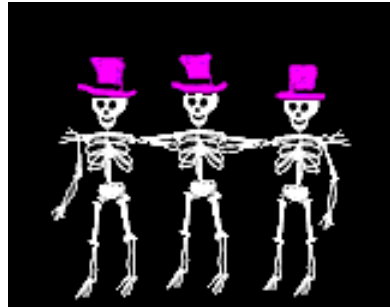
Faster Computer Vs Better Algorithm



Algorithmic improvement more useful than hardware improvement.

E.g. 2^n to n^3

Performance Measurement



Performance Analysis

Paper and pencil.

Don't need a working computer program or even a computer.

Some Uses Of Performance Analysis

- determine practicality of algorithm
- predict run time on large instance
- compare 2 algorithms that have different asymptotic complexity
 - e.g., $O(n)$ and $O(n^2)$

Limitations of Analysis

Doesn't account for constant factors.

but constant factor may dominate

$1000n$ vs n^2

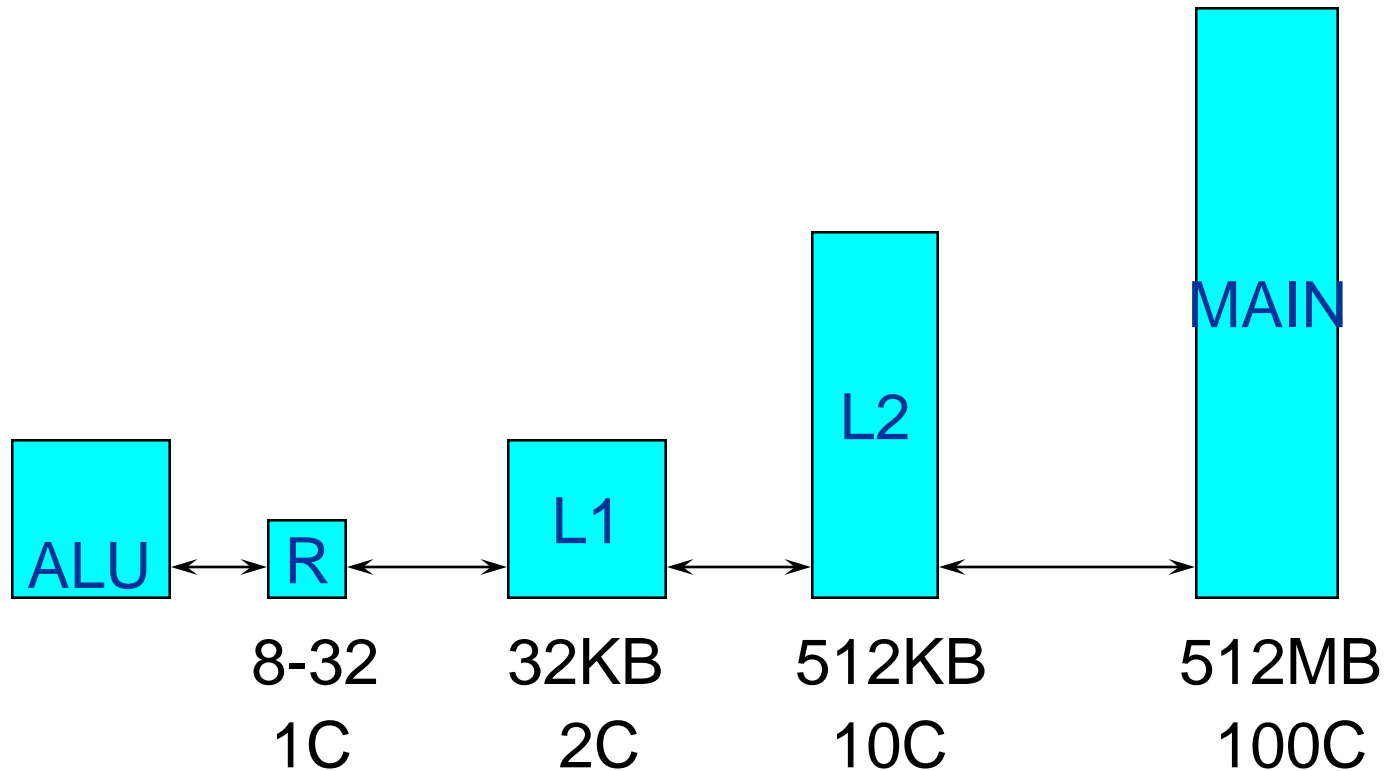
and we are interested only in

$n < 1000$

Limitations of Analysis

Modern computers have a hierarchical memory organization with different access time for memory at different levels of the hierarchy.

Memory Hierarchy



Limitations of Analysis

Our analysis doesn't account for this difference in memory access times.

Programs that do more work may take less time than those that do less work.

Performance Measurement

Measure actual time on an actual computer.

What do we need?

Performance Measurement Needs

programming language

working program

computer

compiler and options to use

Performance Measurement Needs

data to use for measurement

worst-case data

best-case data

average-case data

timing mechanism --- clock



Timing In C++

```
long start, stop;
```

```
time(start); // set start to current time in  
             // hundredths of a second
```

```
// code to be timed comes here
```

```
time(stop); // set stop to current time
```

```
long runTime = stop - start;
```

Shortcoming



Clock accuracy

assume $1/100$ second

Repeat work many times to bring total
time to be $\geq 1/10$ second

Accurate Timing

```
time(start);  
long counter;  
do {  
    counter++;  
    doSomething();  
    time(stop);  
} while (stop - start < 10)  
double elapsedTime = stop - start;  
  
double timeForTask = elapsedTime/counter;
```

Accuracy

Now accuracy is 10%.

first reading may be just about to change to
start + 1

second reading may have just changed to stop

so stop - start is off by 1 unit

Accuracy

first reading may have just changed to
start

second reading may be about to change
to stop + 1

so stop - start is off by 1 unit

Accuracy

Examining remaining cases, we get

$$\text{trueElapsedTime} = \text{stop} - \text{start} \pm 1$$

To ensure 10% accuracy, require

$$\begin{aligned} \text{elapsedTime} &= \text{stop} - \text{start} \\ &\geq 10 \end{aligned}$$

What Went Wrong?

```
time(start);  
long counter;  
do {  
    counter++;  
    insertionSort(a,n);  
    time(stop);  
} while (stop - start < 10)  
double elapsedTime = (stop - start);  
  
double timeToSort = elapsedTime/counter;
```

The Fix

```
time(start);  
long counter;  
do {  
    counter++;  
    // put code to initialize a here  
    insertionSort(a,n);  
    time(stop);  
} while (stop - start < 10)
```

Bad Way To Time

```
do {  
    counter++;  
    time(start);  
    doSomething();  
    time(stop)  
    elapsedTime += stop - start;  
} while (elapsedTime < 10)
```

Time Complexity ,,

Cases

- Worst case
- Best case
- Average case ,,

**** *Worst case and average case analysis is much more useful in practice***

Time Complexity(cont.)

$$„T(P) = c + T_p(I)$$

- c: compile time
- $T_p(I)$: program execution time

Depends on characteristics of instance I „

Predict the growth in run time as the instance characteristics change

Time Complexity (cont.)

Compile time (c)

- Independent of instance characteristics „

Run (execution) time T_p „

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

Time Complexity

Cases

- Worst case
- Best case
- Average case „

Worst case and average case analysis is much more useful in practice

Time Complexity (cont.)

Difficult to determine the exact step counts „

What a step stands for is inexact – e.g. $x := y$
v.s. $x := y + z + (x/y) + \dots$ „

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 as
the instance characteristics change

Space Complexity

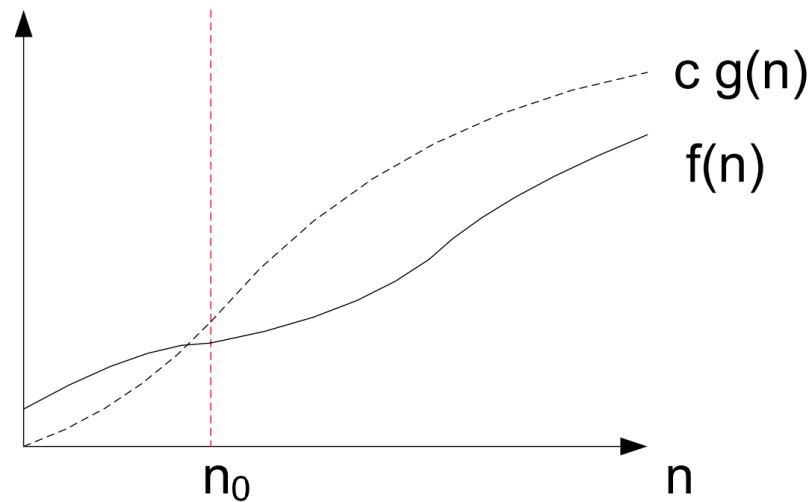
$$S(P) = c + Sp(I)$$

- c : fixed space (instruction, simple variables, constants)
- $Sp(I)$: depends on characteristics of instance I
- Characteristics: number, size, values of I/O associated with I „

If n is the only characteristic, $Sp(I) = Sp(n)$

Asymptotic Notation -Big “oh”

$f(n)=O(g(n))$ iff – \exists a real constant $c>0$
and an integer constant $n_0 \geq 1$, $\exists f(n) \leq$
 $cg(n) \forall n, n \geq n_0$



Asymptotic Notation -Big “oh” (cont.)

Examples

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

$$3n+2 \leq 4n \text{ for all } n \geq 2$$

- $10n^2+4n+2 = O(n^2)$

$$10n^2+4n+2 \leq 11n^2 \text{ for all } n \geq 10$$

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

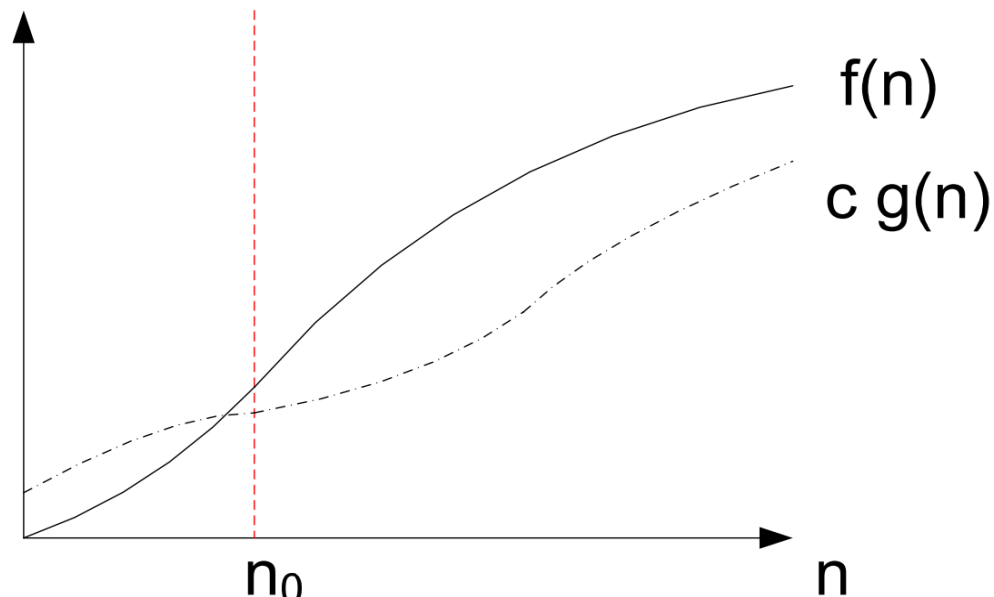
$$3n+2 \leq n^2 \text{ for all } n \geq 4 \text{ (Not tight enough)}$$

- $g(n)$ should be a least upper bound

Asymptotic Notation -Omega

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

- \exists a **real constant** $c > 0$ **and** an **integer constant** $n_0 \geq 1$, \exists
- $f(n) \geq cg(n) \quad \forall n, n \geq n_0$



Asymptotic Notation –Omega (cont.)

Examples

– $3n+3=\Omega(n)$

$3n+3\geq 3n$ for all $n\geq 1$

– $6\cdot 2^n+n^2=\Omega(2^n)$

$6\cdot 2^n+n^2\geq 2^n$ for all $n\geq 1$

– $3n+3=\Omega(1)$

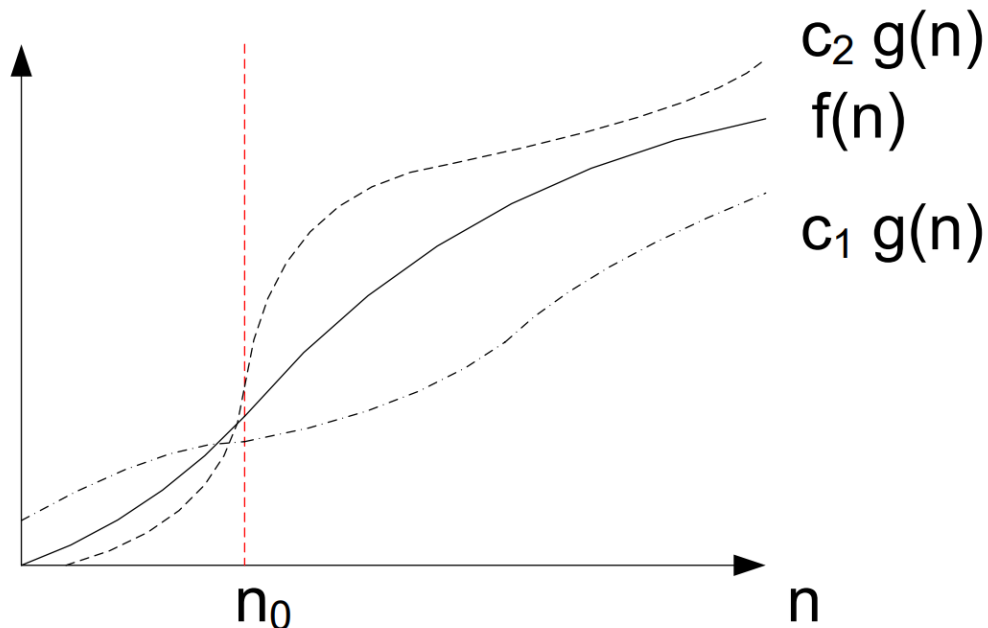
$3n+3\geq 3$ for all $n\geq 1$ „

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

Asymptotic Notation -Theta

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

- \exists two positive real constants $c_1, c_2 > 0$, and an integer constant $n_0 \geq 1$, $\exists c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n, n \geq n_0$



Asymptotic Notation –Theta (cont.)

„Examples

– $3n+2 = \Theta(n)$

$$3n \leq 3n+2 \leq 4n, \text{ for all } n \geq 2$$

– $10n^2+4n+2 = \Theta(n^2)$

$$10n^2 \leq 10n^2+4n+2 \leq 11n^2, \text{ for all } n \geq 5$$

$g(n)$ should be both **lower bound & upper bound**

Some Rules

Rule 1:

- If $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$ Then
 - (a) $T_1(N) + T_2(N) = \max (O(f(N)), O(g(N)))$
 - (b) $T_1(N) \times T_2(N) = O(f(N) \times g(N))$ „

Rule 2:

- If $T(N)$ is a polynomial of degree k , then
 $T(N) = \Theta(N^k)$

Typical Growth Rate

c : constant „

$\log N$: logarithmic „

$\log^2 N$: Log-squared „

N : Linear „

$N \log N$: „

N^2 : Quadratic „

N^3 : Cubic „

2^N : Exponential

