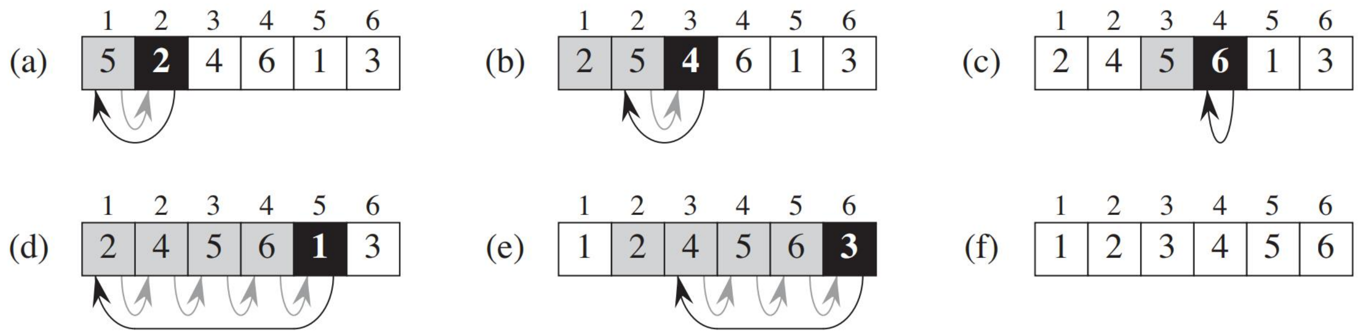
CSC3100: Growth of Functions

Kaiming Shen

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Running Time of Insertion Sort




```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
          sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Running Time of Insertion Sort

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted
              sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

- Cost c_i denotes the running time of line i
- If line i executes n times, it contributes $c_i n$ to the total running time

Running Time of Insertion Sort

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
            sequence A[1 . . j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

- The running time varies from sequence to sequence.
- The best case is that the sequence is already sorted. (Why?)

Running Time of Insertion Sort

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$\quad key = A[j]$	c_2	$n - 1$
$\quad$ // Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n - 1$
$\quad i = j - 1$	c_4	$n - 1$
$\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$n - 1$
$\qquad A[i+1] = A[i]$	c_6	
$\qquad i = i - 1$	c_7	
$\quad A[i+1] = key$	c_8	$n - 1$

- A **for** (or **while**) loop is executed one time more than the loop body.

Running Time of Insertion Sort

for i = 1 **to** 3
    loop body

i = 1, for loop test -> true -> exec loop body x1
i = 2, for loop test -> true -> exec loop body x2
i = 3, for loop test -> true -> exec loop body x3
i = 4, for loop test -> false -> exit

Hence, the for loop test is executed 4 times, while the loop body is executed 3 times.

Running Time of Insertion Sort

So, the best-case running time is

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\ .
\end{aligned}
$$

We further consider a general case in which the while-loop cannot be ignored.

Running Time of Insertion Sort

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $\quad$ $key = A[j]$	c_2	$n-1$
3 $\quad$ // Insert $A[j]$ into the sorted sequence $A[1 \mathinner{..} j-1]$.	0	$n-1$
4 $\quad$ $i = j - 1$	c_4	$n-1$
5 $\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $\quad\quad$ $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $\quad\quad$ $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $\quad$ $A[i+1] = key$	c_8	$n-1$

- t_j denotes the number of times the while loop test in line 5 is executed for a particular *j.*
- A **while** loop test is also executed one time more than the loop body.

Running Time of Insertion Sort

With the while-loop executed t_j times, the overall running time of Insertion_Sort is

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

Running Time of Insertion Sort

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

- We now consider the worst case so that the running time is the longest.
- Reversely sorted sequence yields the worst case. (Why?)

Running Time of Insertion Sort

The general running time is

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

The worst-case running time is

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

Running Time of Insertion Sort

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

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

Running Time of Insertion Sort

- Running time:
  - Best case: $pn+q$
  - Worst case: an^2+bn+c

- The extra precision is not usually worth the effort of computing it.

- For large enough inputs n, the multiplicative constants and lower-order terms of an exact running time are dominated by the effects of the input size itself.

Θ Notation

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

$f(n)$ is "sandwiched" between the lower bound $c_1 g(n)$ and the upper bound $c_2 g(n)$ for sufficiently large n.

Θ Notation

Q: Prove $3n^2 - 6n = \Theta(n^2)$

A: We need to show that

$$c_1 n^2 \leq 3n^2 - 6n \leq c_2 n^2 \text{ , for all } n \geq n_0$$

After simplifying, we obtain

$$c_1 \leq 3 - 6/n \leq c_2$$

We complete proof by choosing c_1=2, c_2=3, and n_0=6.

Big O Notation

$f(n) = O(g(n))$ if there exist positive constant c and n_0 such that $0 \le f(n) \le cg(n)$ for all $n \ge n_0$.

$f(n)$ is bounded from above by $cg(n)$ for sufficiently large n.

Big O Notation

Q: Prove $3n^2 - 6n = O(n^3)$

A: We need to show that

$$0 \leq 3n^2 - 6n \leq cn^3 \text{ , for all } n \geq n_0$$

which can be rewritten as

$$0 \leq 3 - 6/n \leq cn$$

We can choose c=3 and n_0=6.

Big O Notation

Other functions in $O(n^2)$:

- n^2
- n
- $n/1000$
- $n^{1.99999}$
- $n^2/lg(lg(lg(n)))$
- n^2+n
- $n^2+1000n$
- $1000n^2+1000n$

Big Ω Notation

$f(n) = \Omega(g(n))$ if there exist positive constant c and n_0 such that $0 \le cg(n) \le f(n)$, for all $n \ge n_0$.

$f(n)$ is bounded from below by $cg(n)$ for sufficiently large n.

Big Ω Notation

Q: Prove $3n^2 - 6n = \Omega(n)$

A: We need to show that

$$cn \leq 3n^2 - 6n \text{ , for all } n \geq n_0$$

which can be rewritten as

$$0 \leq 3 - (6+c)/n$$

We can choose c=1 and n_0=7.

Big Ω Notation

Other functions in $\Omega(n^2)$:

- n^2
- n^2+n
- $n^2+1000n$
- n^3
- $n^{2.00001}$
- $n^2 lg(lg(lg(n)))$
- $n^2 - n$
- $n^2 - 1000n$

Running Time of Insertion Sort

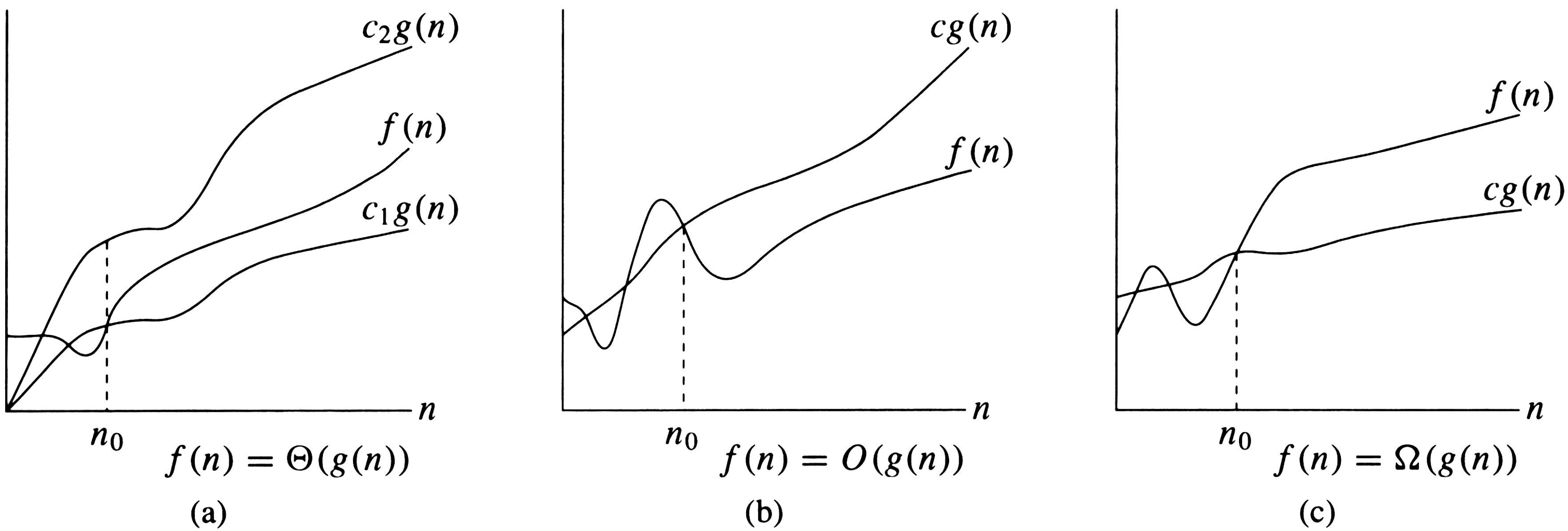
- Running time $T(n)$:
  - Best case: $pn+q$
  - Worst case: an^2+bn+c

- $T(n) = \Theta(n^2)$ in the worst case
- $T(n) \neq \Theta(n^2)$ in general

- $T(n) = \Omega(n)$ in the best case
- $T(n) = \Omega(n)$ in general

- $T(n) = O(n^2)$ in the worst case
- $T(n) = O(n^2)$ in general

Graphing



tight bound upper bound lower bound

Little o Notation

- $f(n) = o(g(n))$ if for any positive constant c there exists n_0 such that $0 \leq f(n) < cg(n)$ for all $n \geq n_0$.

- $f(n) = O(g(n))$ if there exist positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

- $2n = o(n^2)$, $2n^2 \neq o(n^2)$

- $2n = O(n^2)$, $2n^2 = O(n^2)$

Little ω Notation

- $f(n) = \omega(g(n))$ if for any positive constant c there exists n_0 such that $0 \leq cg(n) < f(n)$ for all $n \geq n_0$.
- $f(n) = \Omega(g(n))$ if there exist positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$.
- $2n^2 = \omega(n)$, $2n^2 \neq \omega(n^2)$
- $2n^2 = \Omega(n)$, $2n^2 = \Omega(n^2)$

Limits

- $f(n)=o(g(n))$
- $f(n)<cg(n)$ for sufficiently large n given any $c>0$
- $f(n)/g(n)<c$ for sufficiently large n given any $c>0$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0\,.$$

Limits

- $f(n)=\omega(g(n))$
- $cg(n)<f(n)$ for sufficiently large n given any c>0
- $f(n)/\mathrm{g(n)}>c$ for sufficiently large n given any c>0

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\infty$$

Comparison of Functions

- Function: ω Ω Θ O o
- Real number: $>$ $\geq$ $=$ $\leq$ $<$

Function:	ω	Ω	Θ	O	o
Real number:	$>$	$\geq$	$=$	$\leq$	$<$

Theorem 3.1

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. ■

Properties

Transitivity:

$$
\begin{array}{lll}
f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) & \text{imply} & f(n) = \Theta(h(n)) \,, \\
f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) & \text{imply} & f(n) = O(h(n)) \,, \\
f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) & \text{imply} & f(n) = \Omega(h(n)) \,, \\
f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) & \text{imply} & f(n) = o(h(n)) \,, \\
f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) & \text{imply} & f(n) = \omega(h(n)) \,.
\end{array}
$$

Reflexivity:

$$
\begin{aligned}
f(n) &= \Theta(f(n)) \,, \\
f(n) &= O(f(n)) \,, \\
f(n) &= \Omega(f(n)) \,.
\end{aligned}
$$

$$
\begin{array}{lll}
f(n) = O(g(n)) & \text{is like} & a \le b \,, \\
f(n) = \Omega(g(n)) & \text{is like} & a \ge b \,, \\
f(n) = \Theta(g(n)) & \text{is like} & a = b \,, \\
f(n) = o(g(n)) & \text{is like} & a < b \,, \\
f(n) = \omega(g(n)) & \text{is like} & a > b \,.
\end{array}
$$

Properties

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)) .$$

Transpose symmetry:

$$f(n) = O(g(n)) \quad \text{if and only if} \quad g(n) = \Omega(f(n)) ,$$
$$f(n) = o(g(n)) \quad \text{if and only if} \quad g(n) = \omega(f(n)) .$$

$$\begin{aligned}
f(n) &= O(g(n)) && \text{is like} && a \le b , \\
f(n) &= \Omega(g(n)) && \text{is like} && a \ge b , \\
f(n) &= \Theta(g(n)) && \text{is like} && a = b , \\
f(n) &= o(g(n)) && \text{is like} && a < b , \\
f(n) &= \omega(g(n)) && \text{is like} && a > b .
\end{aligned}$$

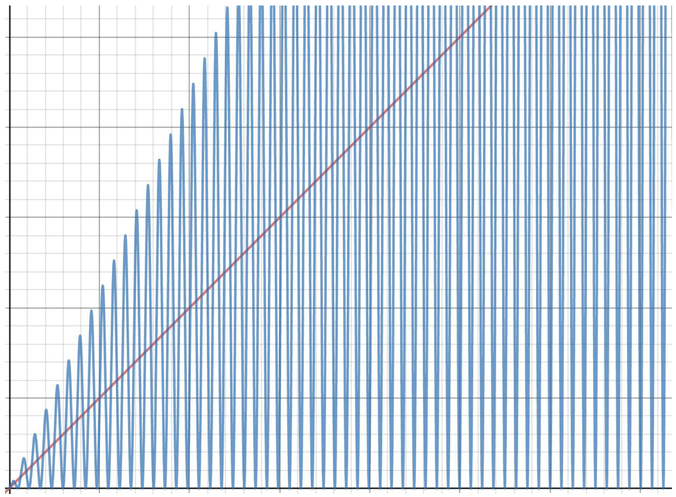
Properties

Any two real numbers are comparable:

Trichotomy: For any two real numbers a and b, exactly one of the following must hold: $a < b$, $a = b$, or $a > b$.

But this property does not carry over to asymptotic notation!

Ex. $f(n)=n$ and $g(n)=n^2(1+\sin n)$



Operation Rules

- Rule 1

$$\text{If } T_1(N) = O(f(N)) \text{ and } T_2(N) = O(g(N)), \text{ then}$$

$$\text{(a) } T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$$

$$\text{(b) } T_1(N) * T_2(N) = O(f(N) * g(N))$$

- Example:

  if $T_1(N) = O(N^2)$ and $T_2(N) = O(N)$ then

  (a) $T_1(N) + T_2(N) = O(N^2)$

  (b) $T_1(N) * T_2(N) = O(N^3)$

Operation Rules

- Rule 2

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

- Rule 3

$$\log^k N = O(N) \text{ for any constant k.}$$

This tells that logarithms grow very slowly

Comparing Functions

Function	Name
c	Constant
logN	Logarithmic
$\log^2 N$	Log-squared
N	Linear
NlogN	
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Operation Rules

Rule 4: Condition Statement

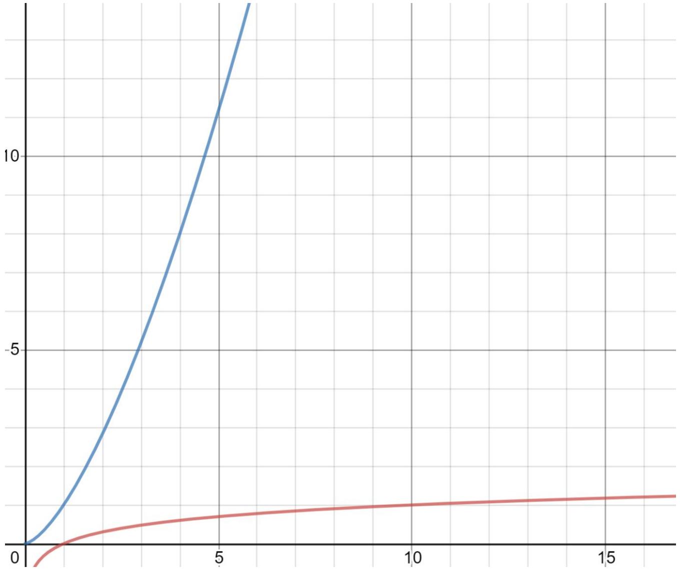
```
if (condition)
        S1
else
        S2
```

If S1 is O(f(n)) and S2 is O(g(n)), then the overall condition statement is O(max(f(n),g(n))).

Exercise #1

If $f(N) = N\log N$, $g(N) = N^{1.5}$, decide which of $f(N)$ and $g(N)$ grows faster.

- $N\log N$ vs $N^{1.5}$
- Divide by N
- $\log N$ vs $N^{0.5}$



Exercise #2

For loops

```
for (i=0;i<N;i++)
    k++
```

O(N)

Nested for loops

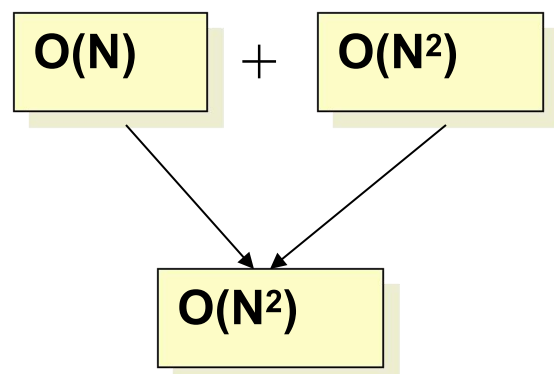
```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        k++
```

$O(N^2)$

Exercise #3

Consecutive Statements

```
for (i=0;i<N;i++)
    A[i] = 0;
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    A[I] += A[j]+i+j;
```



Summary

- Compute the exact running time for insertion sort.
- Asymptotic notation: theta, big O, big Omega, little o, little omega
- Asymptotic running time of algorithm