

# Ch4: Analyzing Insertion-Sort

305234

Algorithm Analysis and Design

Jiraporn Pooksook  
Naresuan University

# Analyze the Correctness of Algorithm

- Using Loop Variants

```
for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[i+1] = A[ i ]
    i = i - 1
  A[i+1]=key
```

# Loop invariants with insertion-sort

A loop invariant =  
all elements in  $A[1 \dots j - 1]$  are in sorted order.

initialization

```
for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[i+1] = A[ i ]
    i = i - 1
  A[i+1]=key
```

mainte-  
nance

termination

# Loop invariants with insertion-sort

A loop invariant = all elements in  $A[1 \dots j - 1]$  are in sorted order.

Initialization

Input = [9,5,7,4,2]

j	key	A[1 to j-1] (before)	i	A[i] > key	A[1 to j-1] (after)	A[1... n]
2	5	[9] is sorted	1	9 > 5		[9,9,7,4,2]
			0		[9] is sorted	[5,9,7,4,2]
3	7	[5,9] is sorted	2	9 > 7		[5,9,9,4,2]
			1	5 > 7 (False)	[5,9] is sorted	[5,7,9,4,2]
4	4	[5,7,9] is sorted	3	9 > 4		[5,7,9,9,2]
			2	7 > 4		[5,7,7,9,2]
			1	5 > 4		[5,5,7,9,2]
			0		[5,5,7] is sorted	[4,5,7,9,2]
5	2	[4,5,7,9] is sorted	4			

## Loop invariants with insertion-sort (Cont.)

A loop invariant = all elements in  $A[1 \dots j - 1]$  are in sorted order.

Input from  $j = 4$  is  $[4, 5, 7, 9, 2]$

$j$	key	$A[1 \text{ to } j-1]$ (before)	$i$	$A[i] > \text{key}$	$A[1 \text{ to } j-1]$ (after)	$A[1 \dots n]$
5	2	$[4, 5, 7, 9]$ is sorted	4	$9 > 2$		$[4, 5, 7, 9, 2]$
			3	$7 > 2$		$[4, 5, 7, 2, 9]$
			2	$5 > 2$		$[4, 5, 2, 7, 9]$
			1	$4 > 2$		$[4, 2, 5, 7, 9]$
			0			$[2, 4, 5, 7, 9]$
6		$[2, 4, 5, 7, 9]$ is sorted				

Termination

# Loop invariants with insertion-sort

theoretically

loop invariant = before running loop  $j$ , all elements in  $A[1 \dots j - 1]$  are in sorted order.

## Initialization:

Before running loop 2, all elements in  $A[1 \dots 1]$  are sorted. (True!!)

## Maintenance:

If before running loop  $j$ , all elements in  $A[1 \dots j - 1]$  are sorted.

then after running loop  $j$ , if  $A[i] > A[j]$  for  $i < j$  then  $A[j]$  will be inserted before position  $i$

and  $A[1 \dots i - 1]$  are sorted where  $A[1 \dots i - 1] < A[j] < A[i]$ .

when we found  $A[i] > A[j]$  for  $i < j$  then  $A[i + 1] = A[j]$  where

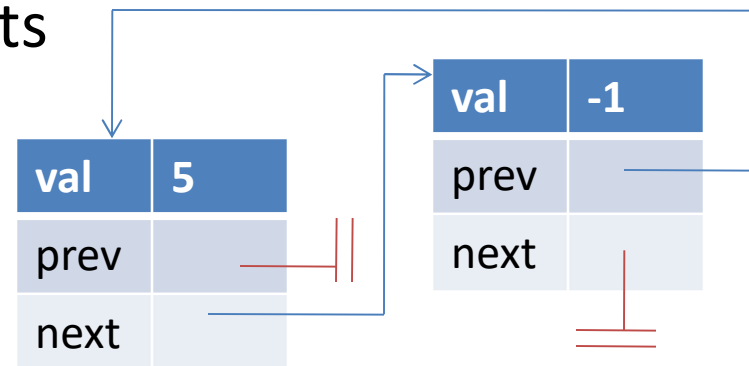
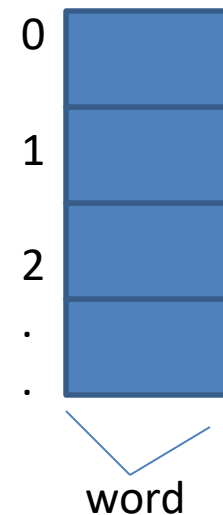
$A[1 \dots i]$  are sorted. Hence  $A[1 \dots j]$  are sorted.

Hence before running loop  $j + 1$ ,  $A[1 \dots j]$  are sorted. (True!!)

**Termination:** at starting of loop  $n + 1$ ,  $A[1 \dots n]$  are sorted (True!!)

# Model of Computation

- Specifies what operations an algorithm is allowed and cost of each operation.
- Random Access Machine (RAM)
  - Modeled by a big array
  - In  $O(1)$  time can load words, do  $O(1)$  computations and store  $O(1)$  words
- Pointer Machine
  - Dynamically allocated objects
  - An object has fields
  - 1 field = word or a pointer



# **THE RUNNING TIME OF ALGORITHM**



# Analyze the Running time of Algorithm

- Name each line of codes.

Line	
$C_1$	for j=2 to length[A]
$C_2$	do key = A[ j ]
$C_3$	
$C_4$	i = j - 1
$C_5$	while i > 0 and A[ i ] > key
$C_6$	do A[i+1] = A[ i ]
$C_7$	i = i - 1
$C_8$	A[i+1]=key

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[i+1] = A[ i ]
    i = i - 1

  A[i+1]=key
    
```

เริ่มจาก 2 ถึง n แสดงว่าลูป  
 นี้รัน n-1 รอบ  
 บวกเพิ่มอีก 1 ตอนที่ต้องเช็ค  
 เงื่อนไขจะเข้าลูปรอบสุดท้าย

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
0	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n t_j - 1$
$C_7$	$\sum_{j=2}^n t_j - 1$
$C_8$	$n-1$

n is the size of  
input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
  do key = A[ j ]
    for i=j-1 to 1
      while A[i] > key
        swap A[i] and A[i-1]
        A[i] = A[i-1]
        A[i-1] = key
    
```

อยู่ในลูปแล้ว(ซึ่งลูปรัน  $n-1$  รอบ) ดังนั้น บรรทัดนี้จะรัน  $n-1$  รอบ

เริ่มจาก 2 ถึง  $n$  แสดงว่าลูปนี้รัน  $n-1$  รอบ  
 บวกเพิ่มอีก 1 ตอนที่ต้องเช็คเงื่อนไขจะเข้าลูปรอบสุดท้าย

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
0	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n t_j - 1$
$C_7$	$\sum_{j=2}^n t_j - 1$
$C_8$	$n-1$

$n$  is the size of input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    A[i+1] = A[i]
    i = i - 1
  A[i+1] = key
  
```

เริ่มจาก 2 ถึง n แสดงว่าลูป  
นี้รัน n-1 รอบ  
บวกเพิ่มอีก 1 ตอนที่ต้องเช็ค  
เงื่อนไขจะเข้าลูปรอบสุดท้าย

ในโค้ดจริงบรรทัดนี้ไม่ได้  
ทำงานอะไร cost = 0  
และโค้ดนี้อยู่ในลูปที่รัน n-  
1 รอบ

Cost	Times
$C_1$	n
$C_2$	n-1
0	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

n is the size of  
input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[i+1] = A[ i ]
    i = i - 1
  
```

เริ่มจาก 2 ถึง n แสดงว่าลูป  
นี้รัน n-1 รอบ  
บวกเพิ่มอีก 1 ตอนที่ต้องเช็ค  
เงื่อนไขจะเข้าลูปรอบสุดท้าย

อยู่ในลูปแล้ว(ซึ่งลูปรัน n-1  
รอบ) ดังนั้น บรรทัดนี้จะรัน  
n-1 รอบ

Cost	Times
$C_1$	n
$C_2$	n-1
0	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

n is the size of  
input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[ i + 1 ] = A[ i ]
    i = i - 1
  
```

กรณีที่ดีที่สุดคือ  $A[i] < key$  ตลอด ดังนั้น จะรันทั้งหมด  $n-1$  รอบ

เพราะเราไม่รู้ว่าจะถูกรันกี่รอบกันแน่ (เป็นได้ทั้งกรณีที่แย่สุด และดีที่สุด หรือกรณีอื่นๆ) เราจึงให้เวลาในการรัน ของรอบที่  $j$  ใดๆ เป็น  $t_j$

กรณีที่แย่ที่สุดคือ  $A[i] > key$  ตลอด ดังนั้น ทุกๆ ครั้งจะต้องรัน เรียงเลขใหม่หมด เช่น สมมติ  $i = 3$  ดังนั้น ก็จะต้องเทียบ  $A[i]$  กับ  $key$  ทั้งหมด 3 รอบ เพราะว่ามันมากกว่า  $key$  ทุกตัว

$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n t_j - 1$
$C_7$	$\sum_{j=2}^n t_j - 1$
$C_8$	$n-1$

$n$  is the size of input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
    do key = A[ j ]

    i = j - 1
    while i > 0 and A[ i ] > key
        do A[i+1] = A[ i ]
        i = i - 1

    A[i+1]=key
    
```

การสลับตัวเลขนี้จะเกิดขึ้น  
ภายในลูปของ จำนวน  $t_j$   
รอบ ดังนั้นจึงมีค่า เป็น  $t_j - 1$

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
0	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n (t_j - 1)$
$C_7$	$\sum_{j=2}^n (t_j - 1)$
$C_8$	$n-1$

$n$  is the size of  
input

# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
    do key = A[ j ]

    i = j - 1
    while i > 0 and A[ i ] > key
        do A[i+1] = A[ i ]
        i = i - 1

    A[i+1]=key
    
```

โค้ดนี้จะเกิดขึ้นภายในลูป  
ของ จำนวน  $t_j$  รอบ ดังนั้น  
จึงมีค่า เป็น  $t_j - 1$

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
0	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n (t_j - 1)$
$C_7$	$\sum_{j=2}^n (t_j - 1)$
$C_8$	$n-1$

$n$  is the size of  
input



# Analyze the Running time of Algorithm

- Using the growth of functions

```

for j=2 to length[A]
    do key = A[ j ]

    i = j - 1
    while i > 0 and A[ i ] > key
        do A[i+1] = A[ i ]
        i = i - 1

A[i+1]=key
    
```

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
$0$	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n t_j - 1$
$C_7$	$\sum_{j=2}^n t_j - 1$
$C_8$	$n-1$

โค้ดนี้จะเกิดขึ้นภายในลูป 2 ถึง  $n-1$  ดังนั้นจึงมีจำนวนรอบเป็น  $n-1$

$n$  is the size of input

# Analyze the Running time of Algorithm

- Using the growth of functions

```
for j=2 to length[A]
  do key = A[ j ]

  i = j - 1
  while i > 0 and A[ i ] > key
    do A[i+1] = A[ i ]
    i = i - 1

  A[i+1]=key
```

Cost	Times
$C_1$	$n$
$C_2$	$n-1$
0	$n-1$
$C_4$	$n-1$
$C_5$	$\sum_{j=2}^n t_j$
$C_6$	$\sum_{j=2}^n t_j - 1$
$C_7$	$\sum_{j=2}^n t_j - 1$
$C_8$	$n-1$

$n$  is the size of input

# The Running time of insertion-sort

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

The best case: all element are ordered.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$an+b$  is a linear function of  $n$

Note!!

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

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

The worst case: a reversed ordered array.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$an^2+bn+c$  is a quadratic function of  $n$

# Analyze the Running time of Algorithm

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

```

for j=2 to n
  do
    i = j - 1
    while i > 0 and A[ i ] > key
      do A[i+1] = A[ i ]
      i = i - 1
  
```

กรณีที่ดีที่สุด เมื่อตัวเลขเรียงมาหมดแล้ว โค้ดบรรทัด c5,c6,c7 ก็จะไม่ต้องทำอะไรเลย

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_1 + c_2 + c_4 + c_5 + c_8)$$

$c_1$	$n$
$c_2$	$n-1$
$0$	$n-1$
$c_4$	$n-1$
$c_5$	$\sum_{j=2}^n t_j$
$c_6$	$\sum_{j=2}^n t_j - 1$
$c_7$	$\sum_{j=2}^n t_j - 1$
$c_8$	$n-1$

n is the size of input

# Analyze the Running time of Algorithm

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

for j=2 to length[A]

กรณีนี้ที่แย่ที่สุด เมื่อตัวเลขเรียงจากมากไปหาน้อยอยู่ก่อน

```

i = j - 1
while i > 0 and A[ i ] > key
do A[i+1] = A[ i ]
i = i - 1
    
```

$c_1$	$n$
$c_2$	$n-1$
$0$	$n-1$
$c_4$	$n-1$
$c_5$	$\sum_{j=2}^n t_j$

ดังนั้น แต่ละรอบของ j ก็จะต้องเรียงตัวเลขทุกตัว ยกตัวอย่างเช่น  
 ถ้า j = 2 ค่า i = 1 ดังนั้น โค้ดนี้ ก็จะต้องรัน 2 รอบ  
 เมื่อ j=3 , i = 2 ดังนั้น โค้ดนี้ก็จะต้องรัน 3 รอบ  
 ไปเรื่อยๆ จนถึง j = n โค้ดนี้ ก็ต้องรัน n รอบ  
 ดังนั้น เราก็จะได้ว่า ค่า  $t_j$  ก็คือค่า j นั่นเอง

$$\sum_{j=2}^n j \Rightarrow 2 + 3 + 4 + \dots + n$$

$$\sum_{j=1}^n j \Rightarrow 1 + 2 + 3 + 4 + \dots + n = \frac{n(n+1)}{2}$$

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

n is the size of input

# The Running time of insertion-sort

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

The best case: all element are ordered.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$an+b$  is a linear function of  $n$

Note!!

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

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

The worst case: a reversed ordered array.

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$an^2+bn+c$  is a quadratic function of  $n$