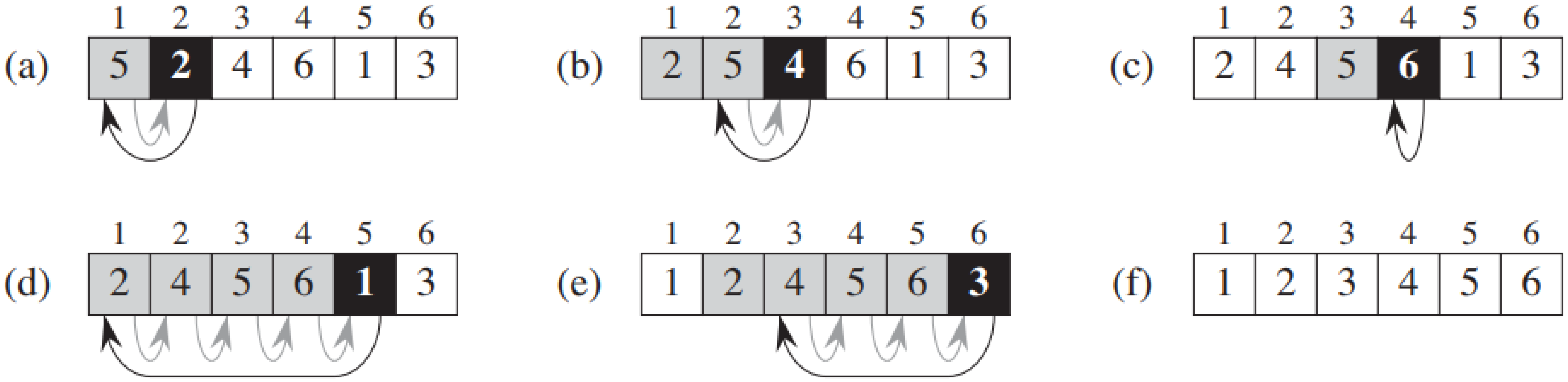
Recap: Insertion Sort



We’re going to analyze its time-efficiency

Insertion Sort.

input size. n=6.

1	2	3	4	5	6
5	2	4	6	1	3

output: ascending. ordered list

outer loop: 2..n.

inner loop:

best case: 1 comparing.

worst case: ~~5-th element.~~
4 comparing

start with 1

5 2 4 6 1 3

2 5 4 6 1 3 iter 1.

2 4 5 6 1 3 iter2.

2 4 5 6 1 3 iter3

1 2 4 5 6 3 iter4

1 2 3 4 5 6 iter5

Time-consuming on input size n

Abstract 1: the size of input n, almost all algorithm run longer on larger input, so the time function is T(n)

Abstract 2: the time spending on each operation vary when using different machine and compiler, we use symbol c_i to denote the time cost. It would make us focus on counting the number of times each operator is executed

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
```

$\boxed{2, 3, \ldots n+1}$

cost	# times
c_1	n
c_2	$n-1$
c_4	$n-1$
c_5	$2+3+\cdots+n$
c_6	$1+2+\cdots n-1$
c_7	$1+2+\cdots n-1$
c_8	$n-1$

worst case (see next page for detail)

Line 5 worst case:

		times	
$j=2$	$i=1, 0$	2	sum
$j=3$	$i=2, 1, 0$	3	$2+3+\dots+n$
$\vdots$			$=\frac{n(n+1)}{2}-1$
$j=n$	$i=n-1, \dots 0$	n	

Line 6 worst case

$j=2.$	$i=1$	1	Sum
$j=3$	$i=2, 1$	2	$1+2+\dots$
$\vdots$			$+(n-1)$
$j=n$	$i=n-1 \dots 1$	$n-1$	

$$=\frac{(n-1)\cdot n}{2}$$

Polynomial

$$T(n) = C_1 \cdot n + C_2(n-1) + C_4(n-1) + C_8(n-1) +$$

$$C_5 \cdot \left(\frac{n(n+1)}{2} - 1\right) + C_6 \cdot \frac{n(n-1)}{2} + C_7 \cdot \frac{n(n-1)}{2}.$$

$$= C_1 \cdot n + (C_2 + C_4 + C_8)(n-1) + C_5 \frac{(n+2)(n-1)}{2}$$

$$+ C_6 \cdot \frac{n(n-1)}{2} + C_7 \frac{n(n-1)}{2}$$

$$= C_1 n + C_2 (n-1) + \frac{n-1}{2}\left[C_5(n+2) + C_6 n + C_7 n\right]$$

$$T(n) = c_1 n + c_2 (n-1) + \frac{n-1}{2} [c_5 (n+2) + c_6 \cdot n + c_7 \cdot n]$$

To count each operation is difficult and unnecessary, apply Approximation

Approximation 1:

After Applying Appromation 1;

count the number of basic operators

$$T(n) = \frac{(n-1)(n+2)}{2} \cdot c_5$$

basic operators: most time-consuming operation in the inner-most loop, e.g. > (in line 5) is the basic operator in insertion-sort algorithm

Approximation 2: ignore the terms in lower order

Apply it $T(n) = c_5 \cdot n^2$

Order of growth: ignore the multiplicative coefficient.

$$T(n) \in O(n^2)$$

why terms in lower-order can be ignored

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

large input size

$T_1(n^2)$: $T(10000) = 100000000$

$T_2(n^2+n)$: $T(10000) = 100010000 \Rightarrow$ 10000 can be ignored

Summary:

Analysing the time-efficiency of an algorithm == find its order of growth.

Path of finding:

Count the number of times basic operation is executed, write down its function f(n) in polynomial, then return its order of growth by keeping the term with highest order and removing the multiplicative coefficient.