When algorithms are implemented,

1. they must always provide the correct answer.
2. they are efficient.
   (1) Computational time (2) Computer memory

We consider the complexity of some algorithms in terms of the number of + and × used.

**Algorithm 1** Matrix Multiplication

A = \([a_{ij}]\) \(m \times k\) and B = \([b_{ij}]\) \(k \times n\)

for \(i = 1 : m\)
   for \(j = 1 : n\)
      \(c_{ij} = 0\)
      for \(q = 1 : k\)
         \(c_{ij} = c_{ij} + a_{iq} b_{qj}\)

return \(C = [c_{ij}]\)
It follows from the algorithm that if two matrices $A$ and $B$ have their size $n \times n$, the number of operations used will be $O(n^3)$, since the actual total number of operations is $n^2(n-1)$.

**Algorithm 2**

$t = 0$
for $i = 1 : 3$
    for $j = 1 : 4$
        $t = t + ij$
    end
end
$t = t + ij$

Since we need the finite number of operations, our big-$O$ estimate will be $O(1)$.

If $i = 1 : m$ and $j = 1 : m$ for any integers $m > 0$, then we can estimate the number of operations, using $O(m^2)$. 
We consider a conventional algorithm (called nested multiplication (or Honers method) for a polynomial at \( x = a \)

\[
P_n(x) = a_{n+1}x^n + a_nx^{n-1} + \cdots + a_2x + a_1.
\]

Its pseudocode is expressed as follows; \( c = \{a_i\}_{i=1}^{n+1} \) is an array which contains all coefficients and \( n \) is the degree of \( P_n \).

**Algorithm 3 Honer’s method**

```plaintext
function y = nested(c, n + 1, x)
    y = a_{n+1}
    for i = n : -1 : 1
        y = y * x + a_i;
    end
end
```

Note that the final value of \( y \) is \( P_n(x) \).

(1) Evaluate \( P_2(x) = 3x^2 + 7x + 1 \) at \( x = 2 \) by (1) by the usual way

(2) Honer’s method

(2) How many \( \times \) and \( + \) are used to evaluate the polynomial at \( x = 2 \)? Answer for both ways.
See the Table 2 (The computer time used by algorithms) in pp.228.

Example 1

1. What is the largest $n$ for which one can solve within one second a problem using an algorithm that requires $f(n)$ bit operations, where each bit operation is carried out in $10^{-9}$ seconds, with these function $f(n)$.
   (1) $\log n$ (2) $n$ (3) $n\log n$
   (4) $n^2$ (5) $2^n$ (6) $n!$

2. How much time does an algorithm using $2^{50}$ operations need if each operation takes these amounts of time?
   (1) $10^{-6}$s (2) $10^{-9}$s (3) $10^{-12}$s