[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.09.08 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

Order of Algorithms

- **Big *O***

For given two functions $f(n)$ and $g(n)$, $g(n) = O(f(n))$ if and only if there exists some positive real constant c and some nonnegative integer N such that $g(n) \leq c \cdot f(n)$ for all $n \geq N$.

➢ We say that $g(n)$ is big O of $f(n)$.

– Example:

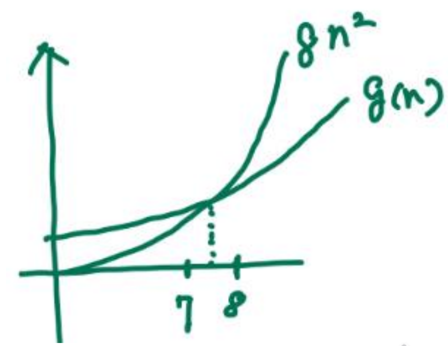
```
x = x + 1;
for (i = 1; i <= n; i++)
  y = y + 2;
for (i = n; i >=1; i--)
  for (j = n; j >= 1; j--)
     z = z + 1;
```

비용 : $g(n) = c_0 + c_1 n + c_2 n^2$

예 : $g(n) = 5 + 6n + 7n^2$

$\rightarrow g(n) \leq \underbrace{8 \cdot n^2}_{c \cdot f(n)}$ for all $n \geq \underbrace{8}_{N}$

$\Rightarrow g(n) = O(n^2)$



Time complexity: $c_0 + c_1 n + c_2 n^2 = O(n^2)$

- **Note 1:** The big *O* puts an asymptotic upper bound on a function.
  - Asymptotic analysis (from Wikipedia)

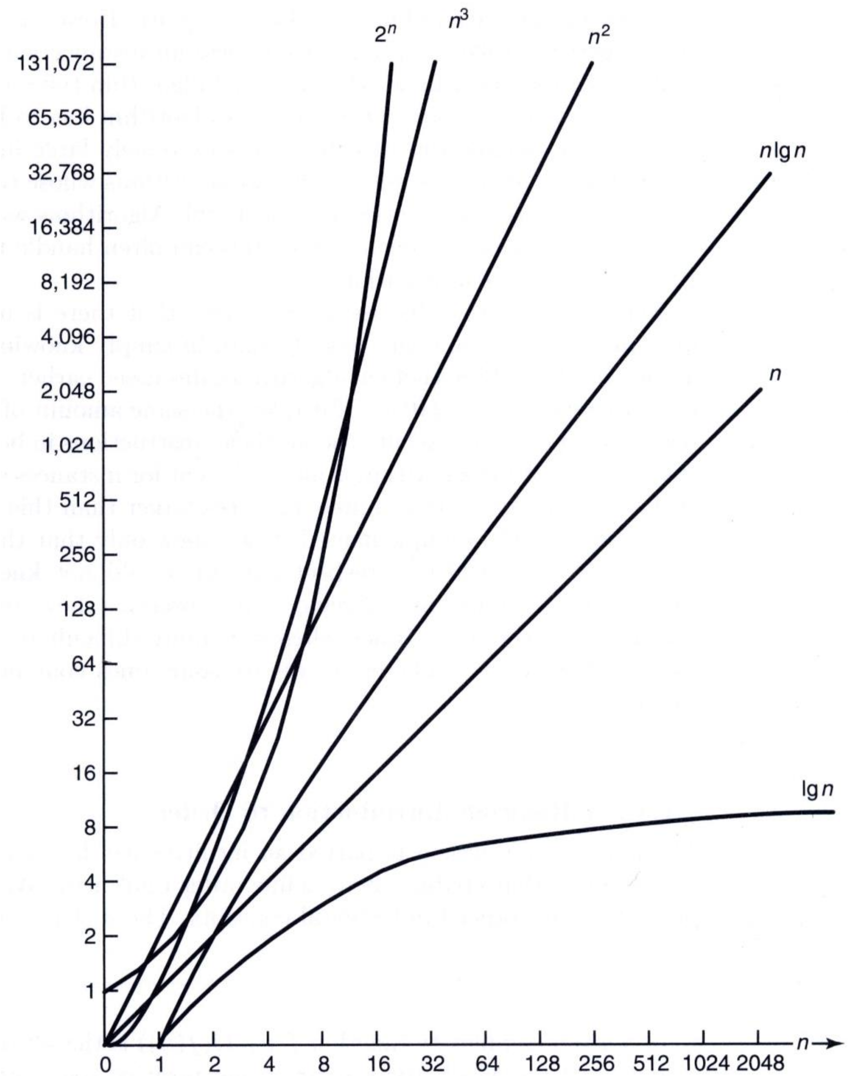
Asymptotic analysis (from Wikipedia)

If f(n) = n^2 + 3n, then as n becomes very large, the term 3n becomes insignificant compared to n^2. The function f(n) is said to be "asymptotically equivalent to n^2, as n → ∞". This is often written symbolically as f(n) ~ n^2, which is read as "f(n) is asymptotic to n^2".

계산 비용이 $0.01n^2$인 알고리즘과 $100n$인 알고리즘 중 어떤 것이 더 "효율적"인가?

  - (Tight) upper bound
    - $37\log n + 0.1n = O(n)$
    - $n^2 + 10n = O(n^2)$
    - $4(\log n)^2 + n\log n + 100n = O(n\log n)$
    - $n^2 + 10n = O(n^{200})$???
    - …

Growth Rates of Some Common Complexity Functions



- **Notes 2:** Given a cost function g(n), how do you find the proper complexity function f(n) such that g(n) = O(f(n))?
  - Suppress lower-order terms and constant factors!
  - Example:

$$* \ 10^3 + 10^3 n + 10^{-3} n^2 = O(n^2)$$

$$\lim_{n\to\infty} \frac{n^2}{n} = \infty$$

$$* \ 5n\log_3 n + 3(\log_2 n)^2 + n + 6n^2 = O(n^2)$$

$$\lim_{n\to\infty} \frac{n}{\log_e n} = \lim_{n\to\infty} n = \infty$$

$$* \ 3(\log_2 n)^2 + 0.1n = O(\ ?\ )$$

$$\boxed{\begin{array}{l} 2^{n+5} = O(2^n) \ ??? \\ 2^{5n} = O(2^n) \ ??? \end{array}}$$

Comparing Orders of Growth

- How do you compare orders of growth of two functions?
  - One possible way is to compute the limit of the ratio of two functions in question.

$$\boxed{x = \lim_{n\to\infty} \frac{f_1(n)}{f_2(n)}}$$

- If $x = 0$, f_1 has a smaller order of growth than f_2.
- If $x = c$, f_1 has the same order of growth as f_2.
- If $x = \infty$, f_1 has a larger order of growth than f_2.

  - Ex. 1: $\log_2 n$ vs $\sqrt{n}$

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n\to\infty} \frac{(\log_2 e)\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = ?$$

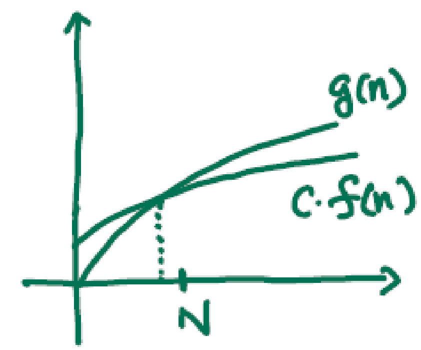
  - Ex. 2: $n!$ vs 2^n

$$\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n}(\frac{n}{e})^n}{2^n} = \lim_{n\to\infty} \sqrt{2\pi n}\frac{n^n}{2^n e^n} = ?$$

$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$ for large value of n: **Stirling's formula**

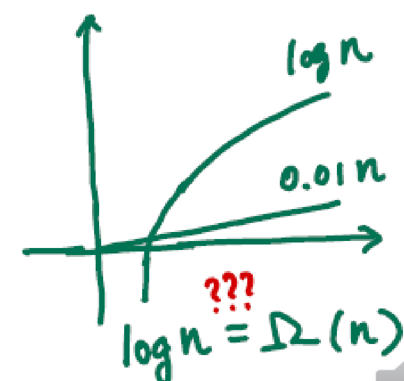
- **Ω (Omega)**

For two given functions $f(n)$ and $g(n)$, $g(n) = \Omega(f(n))$ if and only if there exists some positive real constant c and some nonnegative integer N such $g(n) \geq c \cdot f(n)$ for all $n \geq N$.



  – We say that $g(n)$ is omega of $f(n)$.
  ✓ The Ω puts an asymptotic lower bound on a function.

  – Ex:
    - $37\log n + 0.1n = \Omega(n)$
    - $n^2 + 10n = \Omega(n^2)$
    - $4(\log n)^2 + n\log n + 100n = \Omega(n\log n)$
    - $n^{200} + 10n = \Omega(n^2)$
    - …



- **Θ (Order)**

For given two functions $f(n)$ and $g(n)$, $g(n) = \Theta(f(n))$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$.
That is, $g(n) = \Theta(f(n))$ if and only if there exist some positive real constant c and d and some nonnegative interger N such that, for all $n \geq N$, $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

– We say that $g(n)$ is order of $f(n)$.

– Ex:
  - $37\log n + 0.1n = \Theta(n)$
  - $n^2 + 10n = \Theta(n^2)$
  - $4(\log n)^2 + n\log n + 100n = \Theta(n\log n)$

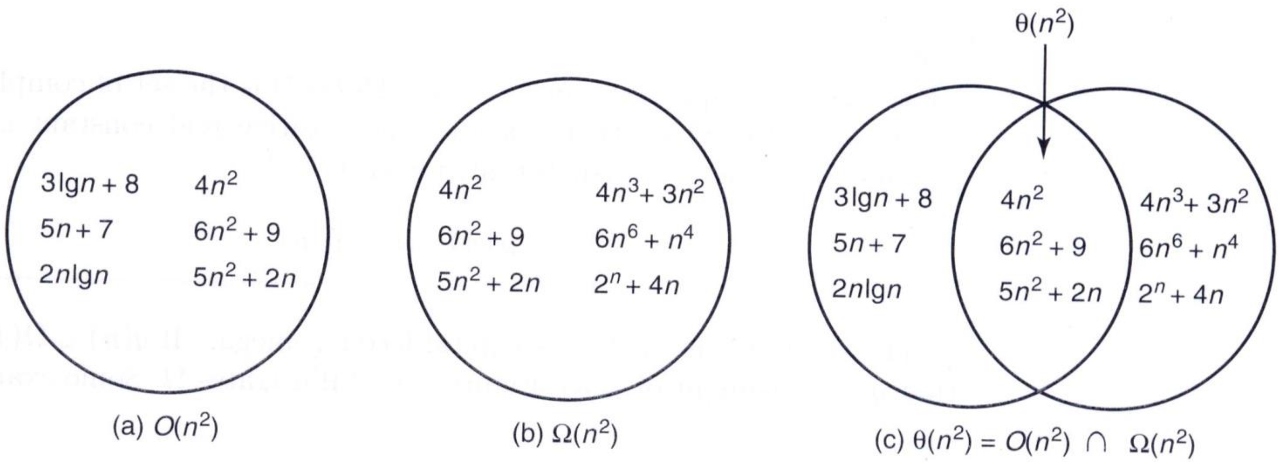
  - $\Theta(1) < \Theta(\log n) < \Theta(n) < \Theta(n\log n) < \Theta(n^2) < \Theta(n^3) < \Theta(n^j) < \Theta(n^k) < \Theta(a^n) < \Theta(b^n) < \Theta(n!)$ $(k > j > 3$ and $b > a > 1)$

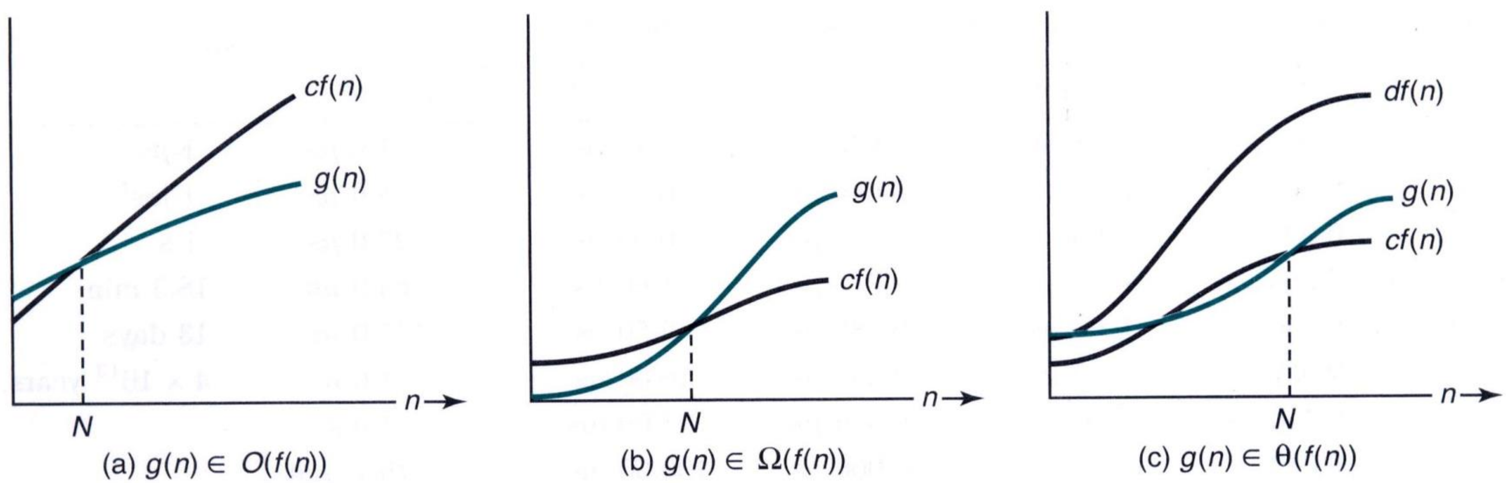
$O(1)$ or $O(c)$: constant

$$\begin{cases} g(n) = 0.000001 \cdot n \\ g(n) = 1000000 \end{cases}$$

[Neapolitan Ex. (pp.42) 19, 24, 26, 28]

Big O, Omega, and Order





[Read Neapolitan Chapter 1.]

Execution Times for Algorithms with the Given Time Complexities

constant

	logarithmic	linear	n-log-n	quadratic	cubic	exponential	factorial
n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$	$< n!$
10	0.003 μs^*	0.01 μs	0.033 μs	0.10 μs	1.0 μs	1 μs	
20	0.004 μs	0.02 μs	0.086 μs	0.40 μs	8.0 μs	1 ms†	
30	0.005 μs	0.03 μs	0.147 μs	0.90 μs	27.0 μs	1 s	
40	0.005 μs	0.04 μs	0.213 μs	1.60 μs	64.0 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.50 μs	125.0 μs	13 days	
10^2	0.007 μs	0.10 μs	0.664 μs	10.00 μs	1.0 ms	4×10^{13} years	
10^3	0.010 μs	1.00 μs	9.966 μs	1.00 ms	1.0 s		
10^4	0.013 μs	10.00 μs	130.000 μs	100.00 ms	16.7 min		
10^5	0.017 μs	0.10 ms	1.670 ms	10.00 s	11.6 days		
10^6	0.020 μs	1.00 ms	19.930 ms	16.70 min	31.7 years		
10^7	0.023 μs	0.01 s	2.660 s	1.16 days	31,709 years		
10^8	0.027 μs	0.10 s	2.660 s	115.70 days	3.17×10^7 years		
10^9	0.030 μs	1.00 s	29.900 s	31.70 years			

Binary search

Merge sort

Bubble sort

Finding all-pairs shortest path

Many intractable combinatorial problems

Finding the closest pair of points

Polynomial

Exponential

Finding the maximum

Merging two sorted lists

From Wikipedia

Notation	Name	Example
$\mathcal{O}(1)$	constant	Determining if a number is even or odd
$\mathcal{O}(\log^* n)$	iterated logarithmic	The `find` algorithm of Hopcroft and Ullman on a disjoint set
$\mathcal{O}(\log n)$	logarithmic	Finding an item in a sorted list with the binary search algorithm
$\mathcal{O}((\log n)^c)$	polylogarithmic	Deciding if n is prime with the AKS primality test
$\mathcal{O}(n^c), 0 < c < 1$	fractional power	searching in a kd-tree
$\mathcal{O}(n)$	linear	Finding an item in an unsorted list
$\mathcal{O}(n \log n)$	linearithmic, loglinear, or quasilinear	Sorting a list with heapsort, computing a FFT
$\mathcal{O}(n^2)$	quadratic	Sorting a list with insertion sort, computing a DFT
$\mathcal{O}(n^c), c > 1$	polynomial, sometimes called algebraic	Finding the shortest path on a weighted digraph with the Floyd-Warshall algorithm
$\mathcal{O}(c^n)$	exponential, sometimes called geometric	Finding the (exact) solution to the traveling salesman problem (under the assumption that P ≠ NP)
$\mathcal{O}(n!)$	factorial, sometimes called combinatorial	Determining if two logical statements are equivalent [1], traveling salesman problem, or any other NP Complete problem via brute-force search
$\mathcal{O}(2^{c^n})$	double exponential	Finding a complete set of associative-commutative unifiers [2]

Worst-Case versus Average-Case Time Complexity

- Expected value (from Wikipedia)

Let X be a random variable with a finite number of finite outcomes $x_1, x_2, \ldots, x_k$ occurring with probabilities $p_1, p_2, \ldots, p_k$, respectively. The **expectation** of X is defined as

$$\mathrm{E}[X] = \sum_{i=1}^{k} x_i\, p_i = x_1 p_1 + x_2 p_2 + \cdots + x_k p_k.$$ [1]

Since the sum of all probabilities p_i is 1 ($p_1 + p_2 + \cdots + p_k = 1$), the expected value is the weighted sum of the x_i values, with the p_i values being the weights.

S_n: the set of all inputs of size n
$c(I)$: the cost of the algorithm on input I
$p(I)$: the probability that input I occurs

- Worst-case complexity

$$T_W(n) = \max\{c(I) \mid I \in S_n\}$$

- Average-case complexity

$$T_A(n) = \Sigma_{I \in S_n} p(I) \cdot c(I)$$

Problem: Find the index of a given value a in a given array $(a_0, a_1, a_2 \cdots, a_{n-1})$. If a does not exist in the array, return -1.

Cost for a linear search algorithm:

Let P_i be the probability such that $a = a_i$.

Then, the average cost is

$$g(n) = 1 \cdot P_0 + 2 \cdot P_1 + 3 \cdot P_2 + \cdots + n \cdot P_{n-1} + n\left(1 - \sum_{k=0}^{n-1} P_k\right)$$

$$= \sum_{k=0}^{n-1} (k+1) P_k + n\left(1 - \sum_{k=0}^{n-1} P_k\right).$$

Ex.1: $n = 10^9$, $P_0 + P_1 + \cdots + P_{10^3} = 1 \Rightarrow g(n) = O(1)$

Ex.2: $n = 10^9$, $P_0 + P_1 + \cdots + P_{n/100} = 1 \Rightarrow g(n) = O(n)$

- **참고:** Quick sort 알고리즘 → Worst-case $O(n^2)$, Average-Case $(n \log n)$

Reviews - Summation

From *WIKIPEDIA: The Free Encyclopedia*
Read http://en.wikipedia.org/wiki/Summation
http://en.wikipedia.org/wiki/List_of_mathematical_series.

- Sums of powers
  - $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
  - $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$
  - $\sum_{i=1}^{n} i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^{n} i\right]^2$
  - $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
  - $\sum_{i=0}^{n} i^s = \frac{(n+1)^{s+1}}{s+1} + \sum_{k=1}^{s} \frac{B_k}{s-k+1}\binom{s}{k}(n+1)^{s-k+1}$

    where B_k is the kth Bernoulli number.
  - $\sum_{i=1}^{\infty} i^{-s} = \prod_{p \text{ prime}} \frac{1}{1-p^{-s}} = \zeta(s)$

    where $\zeta(s)$ is the Reimann zeta function.

- Growth rates
  - $\sum_{i=1}^{n} i^c \in \Theta(n^{c+1})$ for real c greater than -1
  - $\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)$
  - $\sum_{i=1}^{n} c^i \in \Theta(c^n)$ for real c greater than 1
  - $\sum_{i=1}^{n} \log(i)^c \in \Theta(n \cdot \log(n)^c)$ for nonnegative real c
  - $\sum_{i=1}^{n} \log(i)^c \cdot i^d \in \Theta(n^{d+1} \cdot \log(n)^c)$ for nonnegative real c, d
  - $\sum_{i=1}^{n} \log(i)^c \cdot i^d \cdot b^i \in \Theta(n^d \cdot \log(n)^c \cdot b^n)$ for nonnegative real $b > 1$, c, d

Reviews - Run Time Analysis

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    a[i][j] = b[i][j] + c[i][j];
```

```
for (i = 1; i <= N; i++)
  if (i % 2 == 0) a[i] = 1;
  else a[i] = -1;
for (i = 1; i <= N; i++)
  for (j = 1; j <= N; j++)
    a[i][j] = i + j;
```

```
for (i = 1; i <= N; i++) {
  if (i % 2) {
    for (j = 1; j <= N; j++)
        a[i][j] = i + j;
  }
  else {
    for (j = 1; j <= N; j++) {
      a[i][j] = 0;
      for (k = 1; k <= N; k++)
        a[i][j] += k;
    }
  }
}
```

```
x = 0;
for (i = 1; i <= N; i++)
  for (j = 1; j <= i; j++)
    x += i + j;
```

```
x = 0;
for (i = 1; i <= N; i++)
  for (j = 1; j <= i; j++)
    for (k = 1; k <= j; k++)
      x += i + j + k;
```

What if this is i*i?

```
x = 0;
for (i = 1; i <= N; i++)
  for (j = 1; j <= i*i; j++)
    if (j % i == 0)
      for (k = 1; k <= j; k++)
        x++;
```

$O(N^4)$

What is the worst-case time complexity of each loop?

```
// n = 2^k for some positive
//               integer k
for (i = 1; i < N; i++) {
  j = n;
  while (j >= 1) {
    // some O(1) computation
    j = j/2;
  }
}
```

```
// n = 2^k for some positive
//               integer k
i  = n;
while (i >= 1) {
  j = i;
  while (j <= n) {
    // some O(1) computation
    j = 2*j;
  }
  i = i/2;
}
```

```
// float x[n][n+1];
for (i = 0; i <= n-2; i++)
  for (j = i+1; j <= n-1; j++)
    for (k = i; k <= n; k++)
      x[j][k] = x[j][k] - x[i][k]*x[j][i]/x[i][i];
```

Could this be faster?

Magic square

```
// n: odd integer
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    s[i][j] = 0;
s[0][(n-1)/2] = 1;
j = (n-1)/2;
for (key = 2; key <= n*n; key++) {
  k = (i) ? (i-1) : (n-1);
  l = (j) ? (j-1) : (n-1);
  if (s[k][l]) i = (i+1)%n;
  else {
    i = k; j = l;
  }
  s[i][j] = key;
}
```

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

$O(n^2)$

Could this be faster?

```
// compute x^n (n >= 0)
m = n; power = 1; z = x;
while (m > 0) {
  while (!(m%2)) {
    m /= 2; z *= z;
  }
  m--; power *=z;
}
```

$O(\log n)$

What is the worst-case time complexity of each loop?

```
x = x + 1;
for (i = 1; i <= n; i++)
  y = y + 2;
for (i = n; i >=1; i--)
  for (j = n; j >= 1; j--)
     z = z + 1;
```

Time complexity: $c_0 + c_1 n + c_2 n^2 = O(n^2)$

```
c = 0; // n > 0
for (i = 1; i <= n; i++)
  for (j = 1; j <= n; j++)
    for (k = 1; k <= n; k = k*2)
      c += 2;
```

Time complexity: $c(\lfloor \log_2 n \rfloor + 1) * n * n = O(n^2 \log n)$

```
i = 1; j = 1; m = 0; // n > 0
while (j <= n) {
  i++;
  j = j + i;
  m = m + 2;
}
```

Time complexity: $??? = O(\sqrt{n})$

Algorithm Design Example

- **Maximum Subsequence Sum (MSS) Problem**

  Given N (possibly negative) integers $A_0, A_1, \cdots, A_{N-1}$, find the maximum value of $\sum_{k=i}^{j} A_k$ for $0 \leq i \leq j \leq N-1$. (For convenience, the maximum subsequence sum is 0 if all the integers are negative.)

- Example
  - (-2, 11, -4, 13, -5, -2) → MSS = 20

Maximum Subarray Problem
Maximum Positive Sum Subarray Problem

Figure 2.2 Running times of several algorithms for maximum subsequence sum (in seconds)

Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	$N = 10$	0.00103	0.00045	0.00066	0.00034
	$N = 100$	0.47015	0.01112	0.00486	0.00063
	$N = 1{,}000$	448.77	1.1233	0.05843	0.00333
	$N = 10{,}000$	NA	111.13	0.68631	0.03042
	$N = 100{,}000$	NA	NA	8.0113	0.29832

- 최대 부분 수열의 합

길이 n인 정수의 수열 $a_0, a_1, a_2, \cdots, a_{n-1}$이 입력으로 주어져 있다.
여기서 부분 수열 $[i, j]$라는 것은 $a_i, a_{i+1}, a_{i+2}, \cdots, a_j$ 를 말한다.
본 문제는 주어진 수열의 부분 수열의 합, 즉 $\sum_{i \le k \le j} a_k$ 의 최대값을 구하는 문제이다. (이때 주어진 수열의 정수가 모두 음수이면 최대 부분 수열의 합은 0 이라고 간주한다)

예를 들어 다음과 같은 수열이 주어졌을 때,

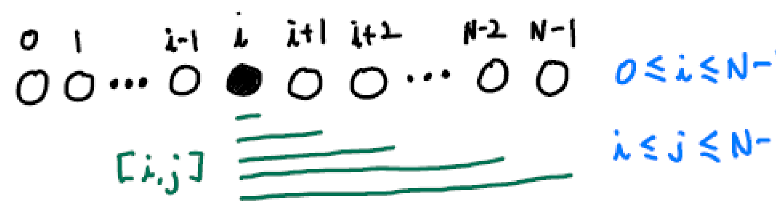
$$+31, -41, +59, +26, -53, +58, +97, -93, -23, +84$$

최대 부분 수열은 $[2,6]$이며 수열의 합은 187 이 된다.

이 문제는 최대 부분 수열의 합을 구하는 것이지만, 앞으로 소개할 알고리즘을 조금만 수정하면 최대 부분 수열도 쉽게 구할 수 있다.

Maximum Subsequence Sum: **Algorithm 1**

- Strategy
  - Enumerate all possibilities one at a time.
  - No efficiency is considered, resulting in a lot of unnecessary computation!




```
int
MaxSubsequenceSum( const int A[ ], int N )
{
    int ThisSum, MaxSum, i, j, k;

    MaxSum = 0;
    for( i = 0; i < N; i++ )
        for( j = i; j < N; j++ )
        {
            ThisSum = 0;
            for( k = i; k <= j; k++ )
                ThisSum += A[ k ];

            if( ThisSum > MaxSum )
                MaxSum = ThisSum;
        }
    return MaxSum;
}
```

Is this for-loop OK for you?

$$\boxed{O(N^3)}$$

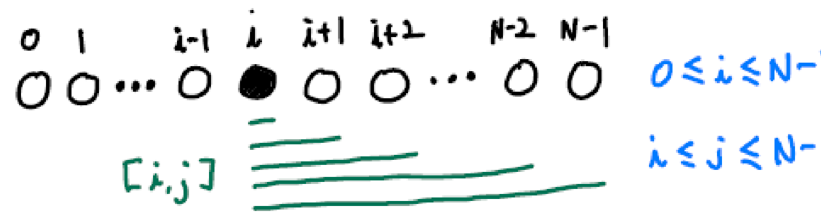
$$\sum_{i=0}^{N-1}\sum_{j=i}^{N-1}\sum_{k=i}^{j} 1 = \frac{N^3+3N^2+2N}{6}$$

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

$$\sum_{k=i}^{j} 1 = j-i+1$$

Maximum Subsequence Sum: **Algorithm 2**

- Strategy
  - Get rid of the inefficiency in the innermost for-loop.
    - Notice that $\sum_{k=i}^{j} A_k = A_j + \sum_{k=i}^{j-1} A_k.$




```
int
MaxSubSequenceSum( const int A[ ], int N )
{
    int ThisSum, MaxSum, i, j;

    MaxSum = 0;
    for( i = 0; i < N; i++ )
    {
        ThisSum = 0;
        for( j = i; j < N; j++ )
        {
            ThisSum += A[ j ];

            if( ThisSum > MaxSum )
                MaxSum = ThisSum;
        }
    }
    return MaxSum;
}
```

$$\sum_{i=0}^{N-1} \sum_{j=i}^{N-1} 1 = ???$$

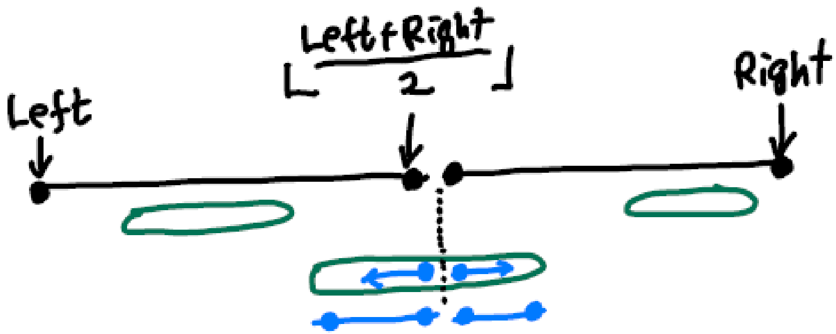
$\boxed{O(N^2)}$

Maximum Subsequence Sum: **Algorithm 3**

- Strategy

  $$\text{Cost: } T(n) = 2T(\frac{n}{2}) + cn, \; T(1) = d$$

  – Use the **Divide-and-Conquer** strategy.
    - The maximum subsequence sum can be in one of three places.



$\boxed{O(N \log N)}$ ← why?

```
static int
MaxSubSum( const int A[ ], int Left, int Right )
{
    int MaxLeftSum, MaxRightSum;
    int MaxLeftBorderSum, MaxRightBorderSum;
    int LeftBorderSum, RightBorderSum;
    int Center, i;

    if( Left == Right )  /* Base Case */
        if( A[ Left ] > 0 )
            return A[ Left ];
        else
            return 0;

    Center = ( Left + Right ) / 2;
    MaxLeftSum = MaxSubSum( A, Left, Center );
    MaxRightSum = MaxSubSum( A, Center + 1, Right );

    MaxLeftBorderSum = 0; LeftBorderSum = 0;
    for( i = Center; i >= Left; i-- )
    {
        LeftBorderSum += A[ i ];
        if( LeftBorderSum > MaxLeftBorderSum )
            MaxLeftBorderSum = LeftBorderSum;
    }

    MaxRightBorderSum = 0; RightBorderSum = 0;
    for( i = Center + 1; i <= Right; i++ )
    {
        RightBorderSum += A[ i ];
        if( RightBorderSum > MaxRightBorderSum )
            MaxRightBorderSum = RightBorderSum;
    }

    return Max3( MaxLeftSum, MaxRightSum,
            MaxLeftBorderSum + MaxRightBorderSum );
}

int
MaxSubsequenceSum( const int A[ ], int N )
{
    return MaxSubSum( A, 0, N - 1 );
}
```