Algorithm Complexity

Jin Hyun Kim
Autumn 2019

Algorithm

- Input: There are zero or more quantities which are externally supplied;
- Output: At least one quantity is produced;
- Definiteness: Each instruction must be clear and unambiguous;
- Finiteness: If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after a finite number of steps;
- Effectiveness: every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be definite, but it must also be feasible

Complexity

- Time Complexity (시간 복잡도) - 알고리즘(연산)이 실행되는 동안에 사용된 **기본적인 연산 횟수를 입력 크기의 함수**로 나타낸다
- Space Complexity (공간 복잡도)

Why to Analyze Algorithm Complexity?

- “Go” game has been very hard to compute so far
- Many computations are not possible to finish for a given time

Time Complexity

- 최악경우 분석(Worst-case Analysis)
  - ‘어떤 입력이 주어지더라도 알고리즘의 수행시간이 얼마 이상은 넘지 않는다’라는 상한(Upper Bound)의 의미
- 평균경우 분석(Average-case Analysis)
  - 입력의 확률 분포를 가정하여 분석하는데, 일반적으로 균등분포(Uniform Distribution)를 가정
- 최선경우 분석(Best-case Analysis)
  - 가장 빠른 수행시간을 분석

Case Analysis

- 집을 나와서 지하철역까지는 5분, 지하철을 타면 학교까지 30분, 강의실까지는 걸어서 10분 걸린다

- **최선경우**: 집을 나와서 5분 후 지하철역에 도착하고, 운이 좋게 바로 열차를 탄 경우를 의미한다. 따라서 최선경우 시간은 5 + 20 + 10 = 35분

- **최악경우**: 열차에 승차하려는 순간, 열차의 문이 닫혀서 다음 열차를 기다려야 하고 다음 열차가 10분 후에 도착한다면, 최악경우는 5 + 10 + 20 + 10 = 45분

Analyzing Algorithm 1

Algorithm 1.1: Sequential Search

Problem: Is the key *x* in the array *S* of *n* keys?

Inputs (parameters): positive integer *n*, array of keys *S* indexed from 1 to *n*, and a key x

Outputs: *location*, the location of *x* in *S* (0 if *x* is not in *S*.)

```
void seqsearch(int n,
               const keytype S [ ],
               keytype x,
               index & location)

{
     location = 1;
     while (location <=  n &&  S[location] != x)
       location ++;
     if (location > n)
        location=0;
}
```

[Foundations of Algorithms Using C++ Pseudocode, Third Edition
by Richard Neapolitan and Kumarss Naimipou]

Analyzing Algorithm 2

Algorithm 1.2: Add Array Members

Problem: Add all the numbers in the array *S* of *n* numbers.

Inputs: positive integer *n*, array of numbers *S* indexed from 1 to *n.*

Outputs: *sum*, the sum of the numbers in *S*.

```
number sum (int n, const number S[ ])
{
      index i;
      number result;
      result = 0;
      for (i = 1; i <=  n; i++)
         result = result = S[i];
      return result;
}
```

[Foundations of Algorithms Using C++ Pseudocode, Third Edition by Richard Neapolitan and Kumarss Naimipou]

Analyzing Algorithm 3

Algorithm 1.3: Exchange Sort

Problem:Sort *n* keys in nondecreasing order.

Inputs: positive integer *n*, array of keys *S* indexed from 1 to *n*..

Outputs: the array *S* containing the keys in nondecreasing order.

```
void exchangesort (int n, keytype S[])
{
   index i, j;
   for (i=1;i<= 1; i++)
      for (j=i+1; j<= n; j++)
         if (S[j] < S[i])
                 exchange S[i] and S[j];
}
```

[Foundations of Algorithms Using C++ Pseudocode, Third Edition
by Richard Neapolitan and Kumarss Naimipou]

Asymptotic Notation

- 알고리즘이 수행하는 **기본 연산 횟수**를 **입력 크기**에 대한 **함수**로 표현
  - O (Big-Oh)
  - Ω (Big-Omega)
  - Θ (Theta)

O (Big-Oh)

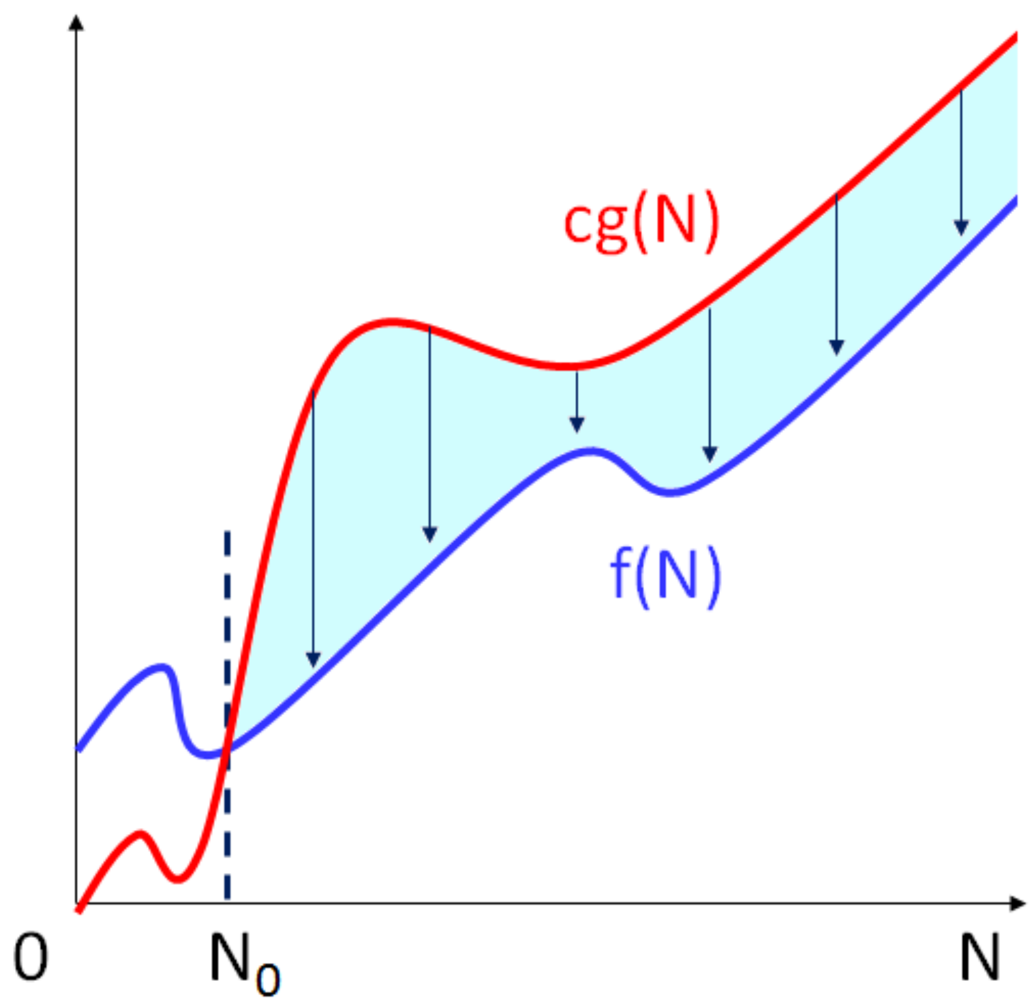
- [**Def: O (Big-Oh)**] 모든 $N \geq N_0$에 대해서, $f(N) \leq cg(N)$이 성립하는 양의 상수 c와 N_0이 존재하면, $f(N) = O(g(N))$이다.

  - O-표기의 의미: N_0과 같거나 큰 모든 N (즉, N_0 이후의 모든 N) 대해서 f(N)이 cg(N)보다 크지 않다는 것

  - $f(N) = O(g(N))$은 N_0 보다 큰 모든 N 대해서 f(N)이 양의 상수를 곱한 g(N)에 미치지 못한다는 뜻

  - g(N)을 f(N)의 상한(Upper Bound)이라고 한다

O (Big-Oh)



$f(N) = O(g(N))$

Example

- f(N) = 2N2 + 3N + 5이면, 양의 상수 c 값을 최고 차항의 계수인 2보다 큰 4를 택하고 g(N) = N^2으로 정하면, 3보다 큰 모든 N에 대해 $2N^2 + 3N + 5 < 4N^2$이 성립, 즉, f(N) = $O(N^2)$

- 물론 2N2 + 3N + 5 = O(N3)도 성립하고, 2N2 + 3N + 5 = O(2N)도 성립한다. 그러나 g(N)을 선택할 때에는 정의를 만족하는 가장 차수가 낮은 함수를 선택하는 것이 바람직함

- f(N) ≤ cg(N)을 만족하는 가장 작은 c값을 찾지 않아도 됨. 왜냐하면 f(N) ≤ cg(N)을 만족하는 양의 상수 c와 N0가 존재하기만 하면 되기 때문

Find Big-Oh

- 주어진 수행시간의 다항식에 대해 O-표기를 찾기 위해 간단한 방법은 다항식에서 최고 차수 항만을 취한 뒤, 그 항의 계수를 제거하여 g(N)을 정한다.

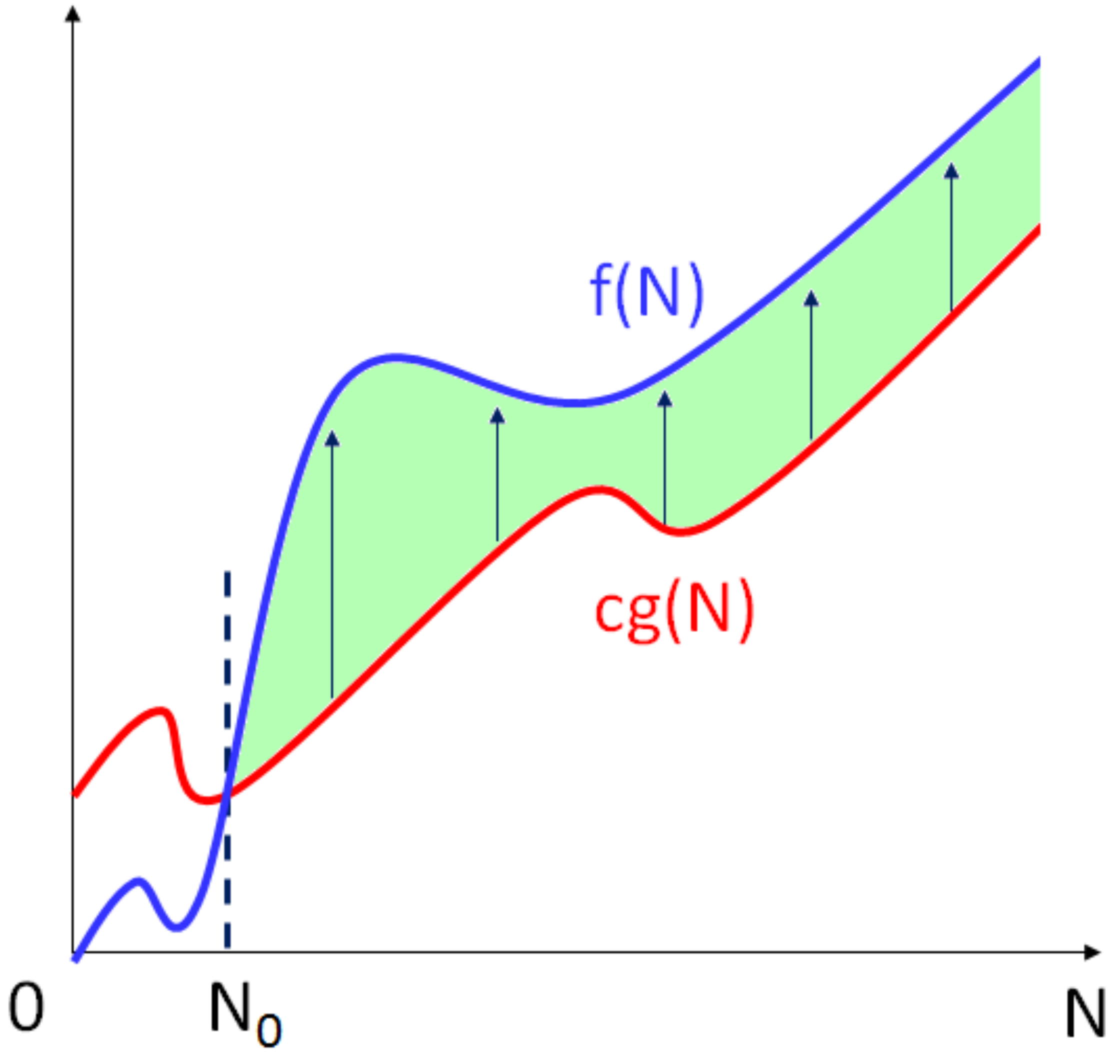
- 예를 들어, $2N^2 + 3N + 5$에서 최고 차수항은 $2N^2$이고, 여기서 계수인 2를 제거하면 N^2이다.

$$2N^2 + 3N + 5 = O(N^2)$$

Ω (Omega)

- [Def: **Ω (Omega)**] 모든 $N \geq N_0$에 대해서 $f(N) \geq cg(N)$이 성립하는 양의 상수 c와N_0이 존재하면, $f(N) = \Omega(g(N))$이다.
  - Ω-표기의 의미는 N_0 보다 큰 모든 N 대해서 f(N)이 cg(N)보다 작지 않다는 것
  - $f(N) = \Omega(g(N))$은 양의 상수를 곱한 g(N)이 f(N)에 미치지 못한다는 뜻
  - g(N)을 f(N)의 하한(Lower Bound)이라고 함

Ω (Omega)



$f(N) \geq cg(N)$

Example

- $f(N) = 2N^2 + 3N + 5$ 일 때, 양의 상수 $c = 1$로 택하고 $g(N) = N^2$으로 정하면, 1보다 큰 모든 N에 대해 $2N^2 + 3N + 5 > N^2$이 성립한다. 따라서 $f(N) = \Omega(N^2)$

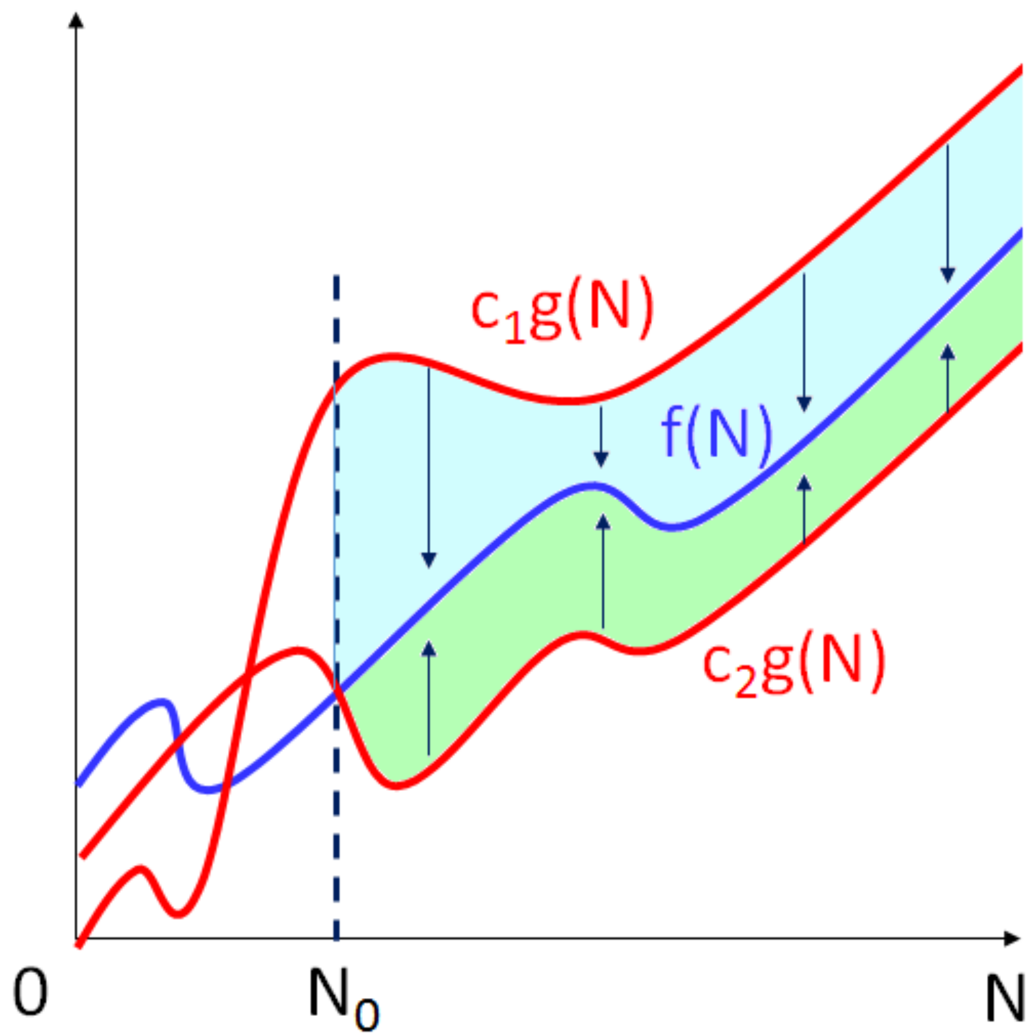
- 물론 $2N^2 + 3N + 5 = \Omega(N)$도 성립하고, $2N^2 + 3N + 5 = \Omega(\log N)$도 성립한다. 그러나g(N)을 선택할 때에는 정의를 만족하는 가장 높은 차수의 함수를 선택하는 것이 바람직함

- $f(N) \geq cg(N)$을 만족하는 가장 작은 양의 c값을 찾아야 하는 것은 아니다. 왜냐하면 $f(N) \geq cg(N)$을 만족하는 양의 상수 c와 N0가 존재하기만 하면 되기 때문

Θ (Theta)

- [Def: **Θ (Theta)**] 모든 $N \ge N0$에 대해서 $c1g(N) \ge f(N) \ge c2g(N)$이 성립하는 양의 상수 c1, c2, N_0가 존재하면, $f(N) = \Theta(g(N))$이다.
  - Θ-표기는 수행시간의 O-표기와 Ω-표기가 동일한 경우에 사용
  - $2N^2 + 3N + 5 = O(N^2)$과 동시에 $2N^2 + 3N + 5 = \Omega(N^2)$ 이므로, $2N^2 + 3N + 5 = \Theta(N^2)$
  - $\Theta(N^2)$은 N2과 $(2N^2 + 3N + 5)$이 유사한 증가율을 가지고 있다는 뜻
  - $2N^2 + 3N + 5 \ne \Theta(N^3)$, $2N^2 + 3N + 5 \ne \Theta(N)$

Θ (Theta)



$c1g(N) \geq f(N) \geq c2g(N)$

Algorithm Complexity Notation

- 알고리즘의 수행시간은 주로 O-표기를 사용하며, 보다 정확히 표현하기 위해 Θ-표기를 사용하기도 한다.
  - O(1) 상수시간(Constant Time)
  - O(logN) 로그(대수)시간(Logarithmic Time)
  - O(N) 선형시간(Linear Time)
  - O(NlogN) 로그선형시간(Log-linear Time)
  - O(N2) 제곱시간(Quadratic Time)
  - O(N3) 세제곱시간(Cubic Time)
  - O(2N) 지수시간(Exponential Time)

Big-Oh

5, 100 — $O(1)$

$O(\log N)$: 5, 100, $2\log N$

$2\log N$

$4N+8$

$O(N)$: 5, 100, $2\log N$, $4N+8$

$N\log N + N$

$3N^2+5N+4$

$O(N^2)$: 5, 100, $2\log N$, $4N+8$, $N\log N + N$, $3N^2+5N+4$

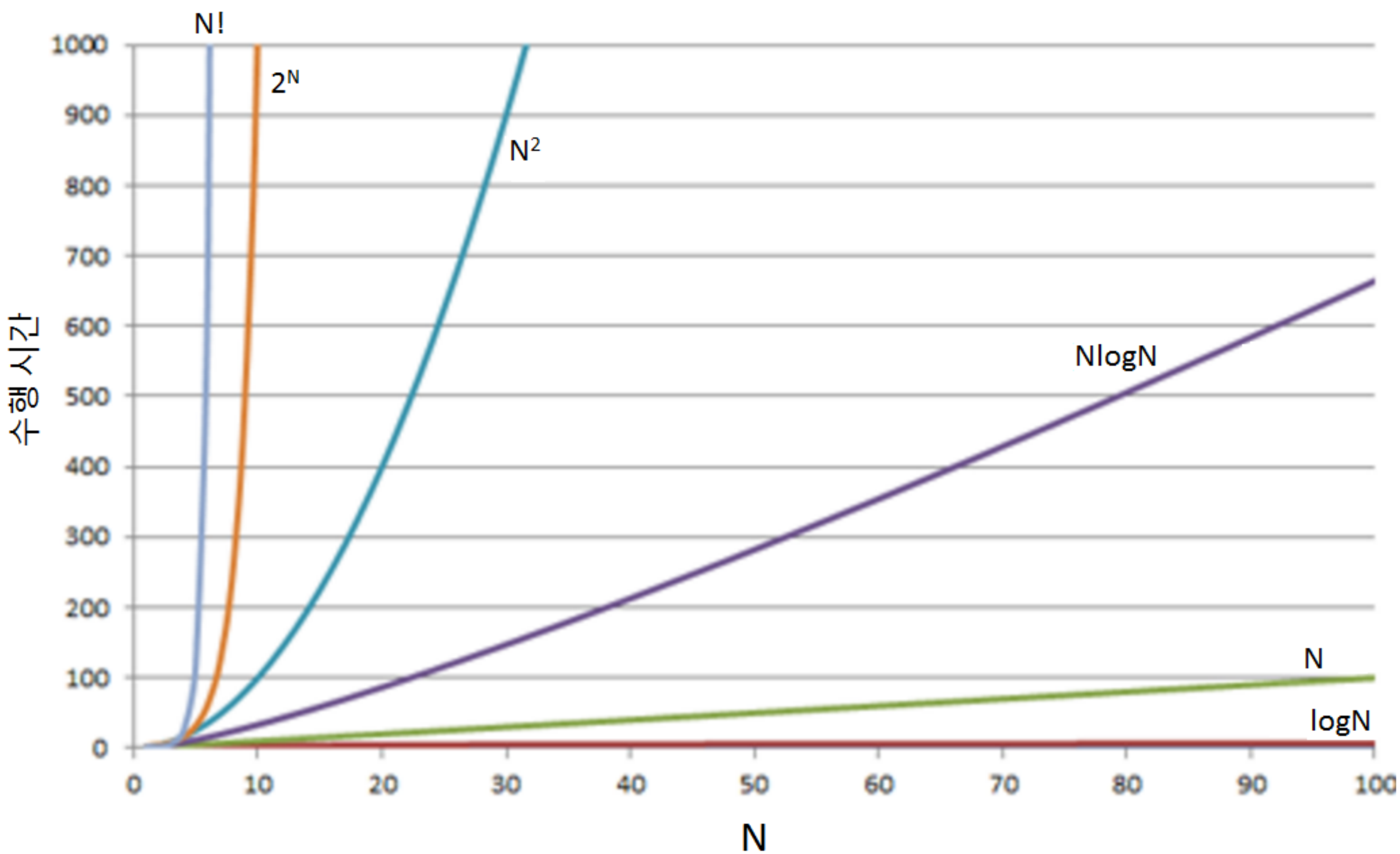
$2N^3+3N^2-N+1$

$O(N^3)$, $O(N^k)$: 5, 100, $2\log N$, $4N+8$, $N\log N + N$, $3N^2+5N+4$, $2N^3+3N^2-N+1$

2^N

$O(2^N)$: 5, 100, $2\log N$, $4N+8$, $N\log N + N$, $3N^2+5N+4$, $2N^3+3N^2-N+1$, 2^N

Increasing Rates



Conclusions

- Data structure is an essential topic for advanced programming
  - It is like fitting a vessel to a bottle
- Algorithm complexity analysis
  - There is a computation that cannot complete on time
- Complexity notation
  - Big-Oh (O)
  - Omega (Ω)
  - Theta (Θ)

Next Time

- Recursion