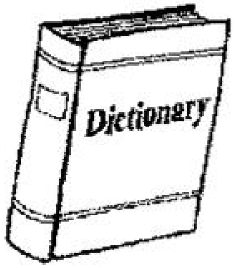
온라인 수업 1주차

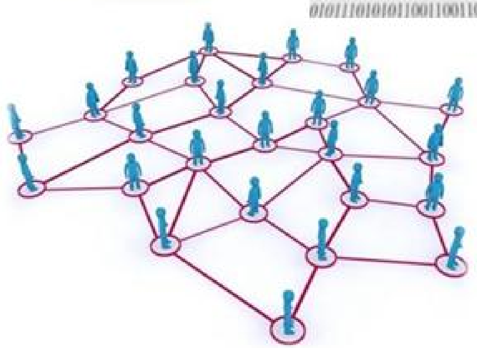
01 CHAPTER

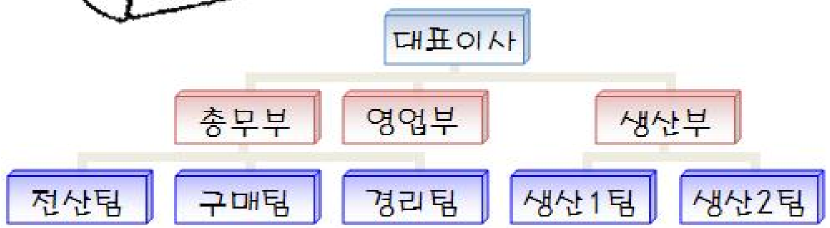
자료구조와 알고리즘

자료구조

- 일상 생활에서 자료를 정리하고 조직화하는 이유는?
  - 사물을 편리하고 효율적으로 사용하기 위함
  - 다양한 자료를 효율적인 규칙에 따라 정리한 예



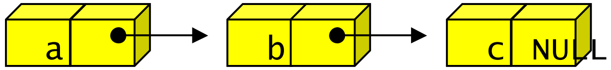


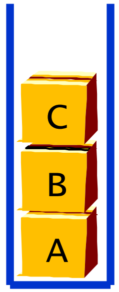


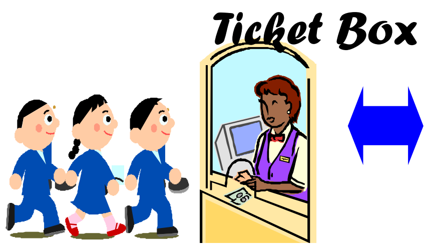
컴퓨터에서의 자료구조

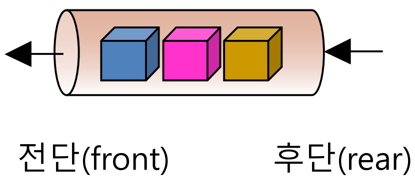
- 자료구조(Data Structure)
  - 컴퓨터에서 자료를 정리하고 조직화하는 다양한 구조

일상생활에서의 예	자료구조
물건을 쌓아두는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색구조
지도	그래프
조직도	트리



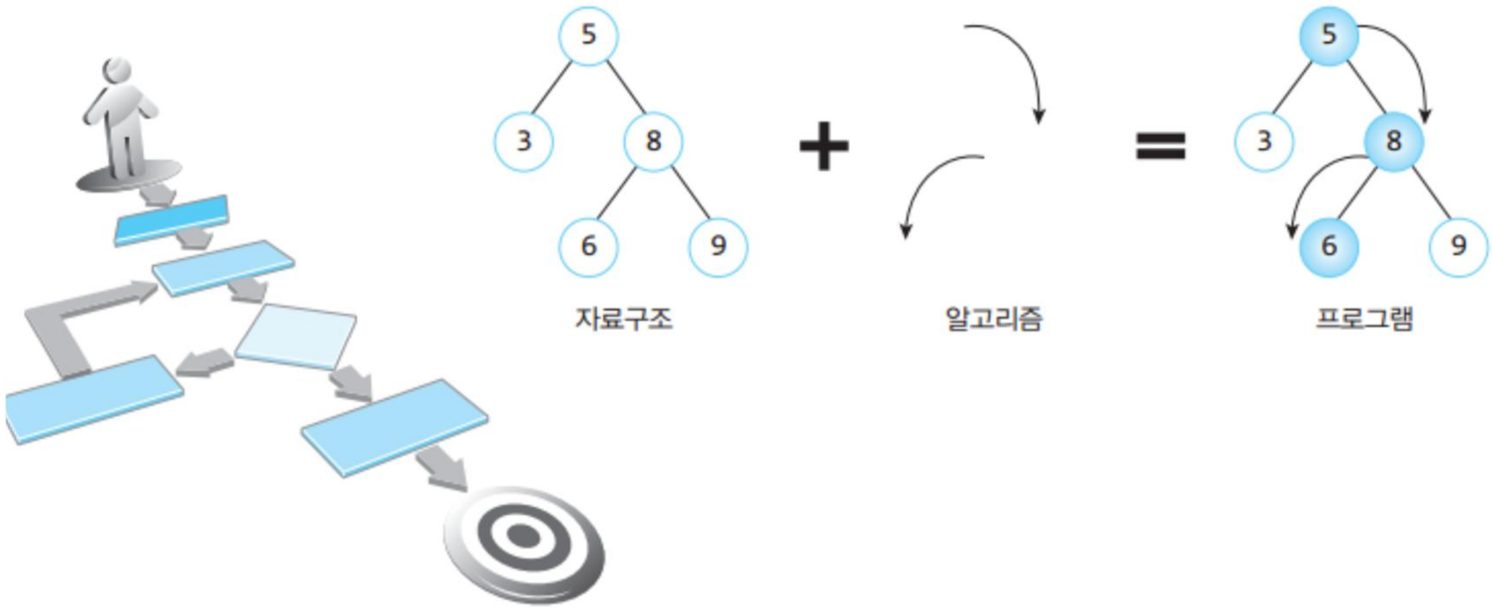






컴퓨터 프로그램의 구성

- 컴퓨터 프로그램은 무엇으로 이루어져 있나?
  - **프로그램** = **자료구조** + **알고리즘**



| 그림 1.3 알고리즘은 문제를 해결하는 절차

- (예) 최대값 탐색 프로그램 = 배열(List)+ 순차탐색

자료구조

score[]

80	70	90	...	30

알고리즘

```
tmp← score[0];
for i←1 to n-1 do
    if score[i]>tmp
        then tmp←score[i];
```

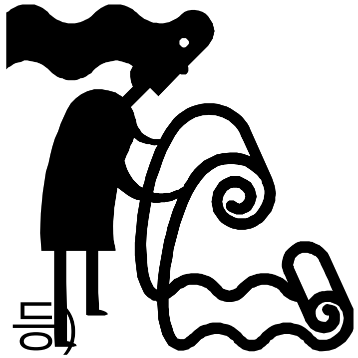
알고리즘

- *컴퓨터로 문제를 풀기 위한 단계적인 절차*
  - 예) 전화번호부에서 특정 사람 이름 찾기

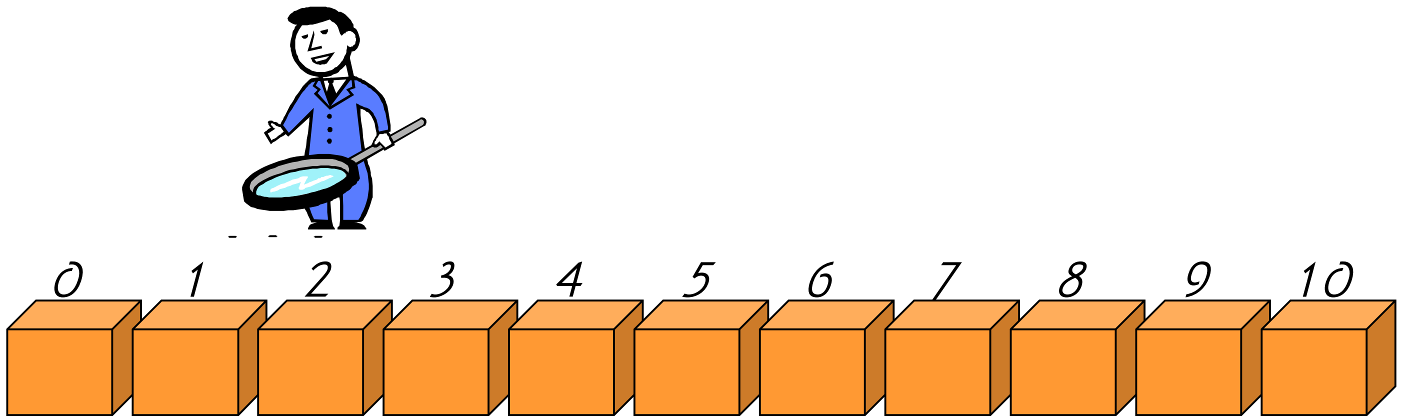
- 알고리즘의 조건
  - 입　력 : 0개 이상의 입력이 존재하여야 한다.
  - 출　력 : 1개 이상의 출력이 존재하여야 한다.
  - 명백성 : 각 명령어의 의미는 모호하지 않고 명확해야 함.
  - 유한성 : 한정된 수의 단계 후에는 반드시 종료되어야 함.
  - 유효성 : 각 명령어들은 실행 가능한 연산이어야 한다.

알고리즘의 기술 방법

- 방법들
  - 영어나 한국어와 같은 자연어
  - 흐름도(flow chart)
  - 유사 코드(pseudo-code)
  - 특정한 프로그래밍 언어 (파이썬, C언어, C++, java 등)



- (예) 배열에서 최대값 찾기 알고리즘



알고리즘의 기술 방법(1)

(1) 자연어로 표기된 알고리즘

- 인간이 읽기가 쉽다.
- 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다.

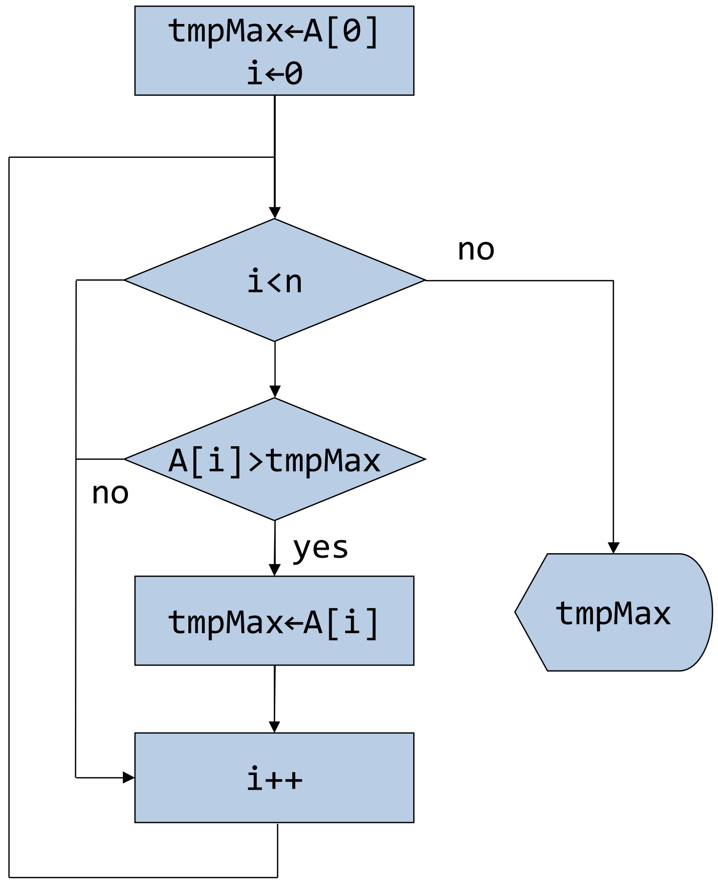
```
ArrayMax(A,n)

1. 배열 A의 첫 번째 요소를 변수 tmp에 복사
2. 배열 A의 다음 요소들을 차례대로 tmp와 비교하면 더 크면 tmp로
   복사
3. 배열 A의 모든 요소를 비교했으면 tmp를 반환
```

알고리즘의 기술 방법(2)

(2) 흐름도로 표기된 알고리즘

– 직관적이고 이해하기 쉬운 알고리즘 기술 방법
– 그러나 복잡한 알고리즘의 경우, 상당히 복잡해짐.



알고리즘의 기술 방법(3)

(3) **유사코드**로 표현된 알고리즘

- 알고리즘의 고수준 기술 방법
- 자연어보다는 더 구조적인 표현 방법
- 프로그래밍 언어보다는 덜 구체적인 표현방법
- 알고리즘 기술에 가장 많이 사용
- 프로그램을 구현할 때의 여러 가지 문제들을 감출 수 있음
- <u>알고리즘의 핵심적인 내용에만 집중</u> 할 수 있음

```
ArrayMax(A,n)

  tmp ← A[0];
  for i←1 to n-1 do
        if tmp < A[i]
        then tmp ← A[i];
  return tmp;
```

대입 연산자가
←임을 유의

알고리즘의 기술 방법(4)

(4) **특정 언어**로 표현된 알고리즘

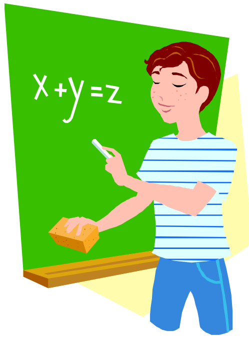
- 알고리즘의 가장 정확한 기술 가능
- 실제 구현시의 많은 구체적인 사항들이 알고리즘의 핵심적인 내용들의 이해를 방해할 수 있음
- 예) 파이썬으로 표기된 알고리즘

```
class Student:
    def __init__(self, name, id):
        self.name = name
        self.id = id
    def get_name(self):
        return self.name
    def get_id(self):
        return self.id

best = Student('Lee', 101)
print(best.get_name())
print(best.get_id())
```

자료형과 추상 자료형

- 추상화란?
  - 어떤 시스템의 간략화 된 기술 또는 명세
  - 시스템의 정말 핵심적인 구조나 동작에만 집중



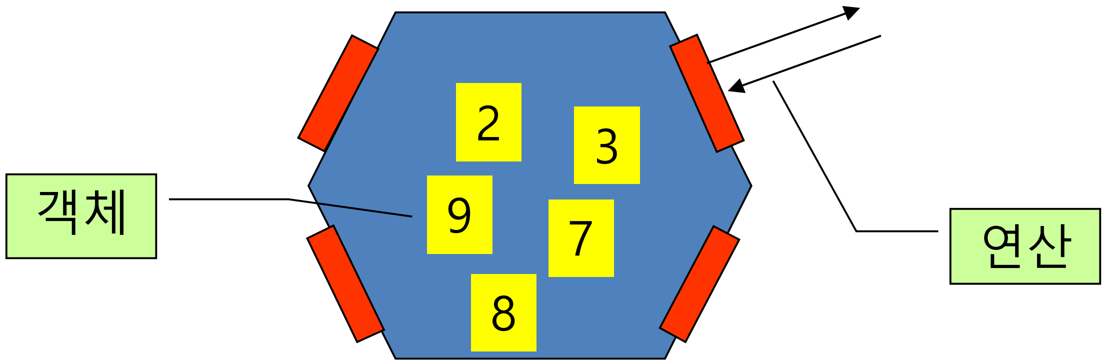
- 자료형(data type)
  - 데이터의 집합과 연산의 집합

  예) int 데이터 타입
  - 데이터: { ...,-2,-1,0,1,2,...}
  - 연산: +, -, /, *, %

- 추상 자료형(ADT: Abstract Data Type)
  - 데이터 타입을 추상적(수학적)으로 정의한 것
  - 데이터나 연산이 **무엇(what)**인가를 정의함
  - 데이터나 연산을 **어떻게(how)** 구현할 것인지는 정의하지 않음

추상 자료형의 정의

- 데이터+연산



- 자연수 ADT

데이터: 1에서 시작하여 INT_MAX까지의 순서화된 정수의 부분 범위

연산:

- add(x,y): x+y가 INT_MAX보다 작으면 x+y를 반환
- distance(x,y): x가 y보다 크면 x-y를 반환하고 작으면 y-x를 반환
- equal(x,y): x와 y가 같은 값이면 TRUE, 아니면 FALSE를 반환
- successor(x): x가 INT_MAX보다 작으면 x+1을 반환한다.

추상데이터타입과 자료구조 관계

추상데이터타입
(ADT)

데이터 + 관련 연산

구현

자료구조

데이터 + 구체화된 관련 연산

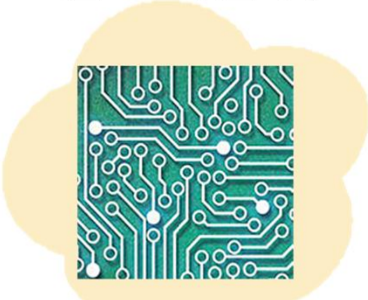
알고리즘의 성능분석

- 알고리즘의 성능 분석 기법
  - **실행 시간을 측정하는 방법**
    - 알고리즘의 실제 실행 시간을 측정하는 것
    - 실제로 구현하는 것이 필요
    - 하드웨어(컴퓨터), 소프트웨어(프로그램언어)등의 환경에 따라 실행 시간(성능)이 달라짐
  - **알고리즘의 복잡도를 분석하는 방법**
    - 직접 구현하지 않고서도 수행 시간을 분석하는 것
    - 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
    - 일반적으로 연산의 횟수는 n의 함수
    - **시간 복잡도 분석** : 수행 시간 분석
    - **공간 복잡도 분석** : 수행시 필요로 하는 메모리 공간 분석

(2) 복잡도 분석

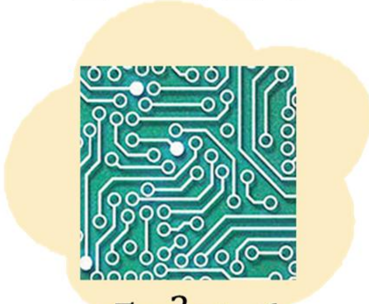
- 시간 복잡도
  - 산술, 대입, 비교, 이동의 기본적인 연산 고려
  - 알고리즘 수행에 필요한 연산의 개수를 계산
  - 입력의 개수 n에 대한 함수->**시간복잡도 함수**, T(n)

알고리즘 **A**



$3n + 2$

알고리즘 **B**



$5n^2 + 6$

복잡도 분석의 예

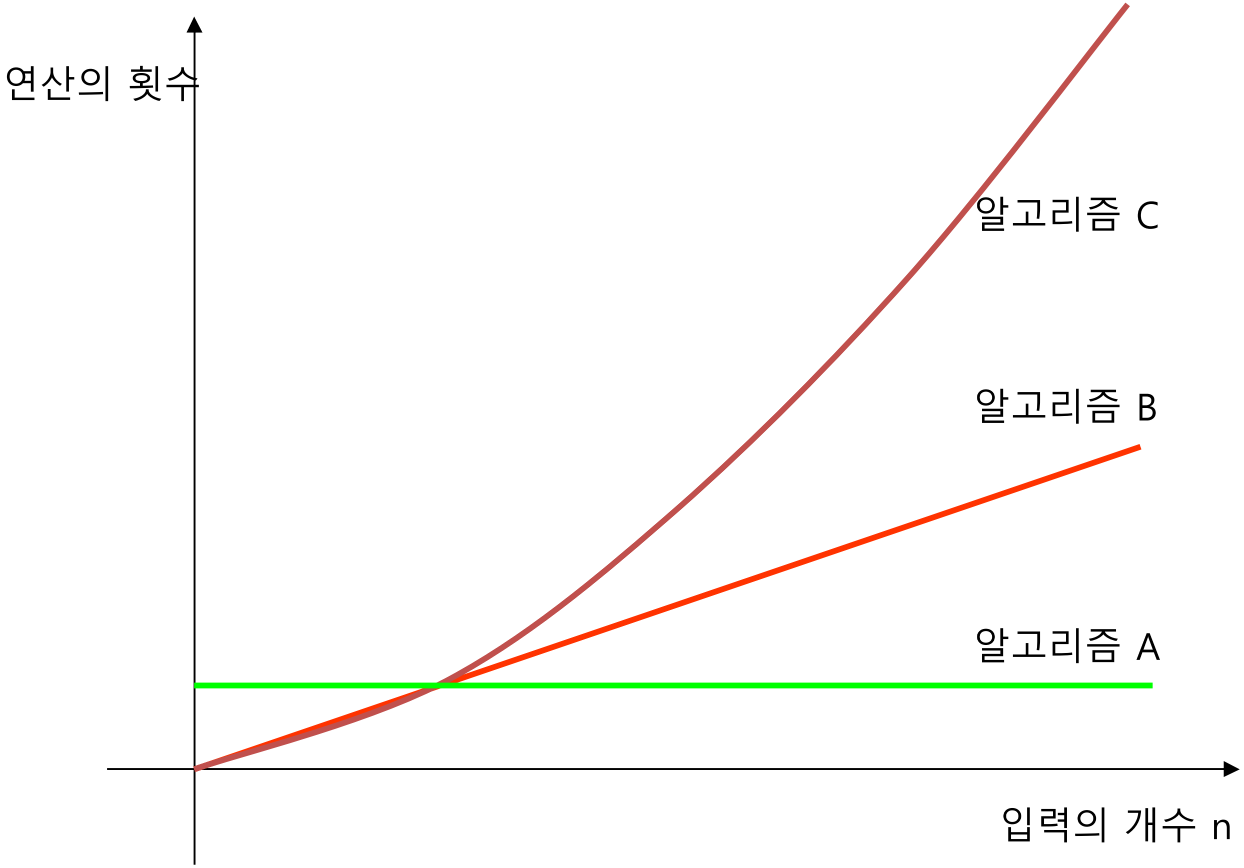
- n을 n번 더하는 문제
  - 각 알고리즘이 수행하는 연산의 개수 계산
  - 단 for 루프 제어 연산은 고려하지 않음

알고리즘 A	알고리즘 B	알고리즘 C

sum ←n*n;	sum ← 0;
	for i ← 1 to n do
	  sum ←sum + n;
		sum ← 0;
		for i←1 to n do
		  for j←1 to n do
		    sum ←sum + 1;

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	n + 1	n*n + 1
덧셈연산		n	n*n
곱셈연산	1		
나눗셈연산			
전체연산수	2	2n + 1	$2n^2 + 1$

연산의 횟수를 그래프로 표현



수행시간의 점근표기법

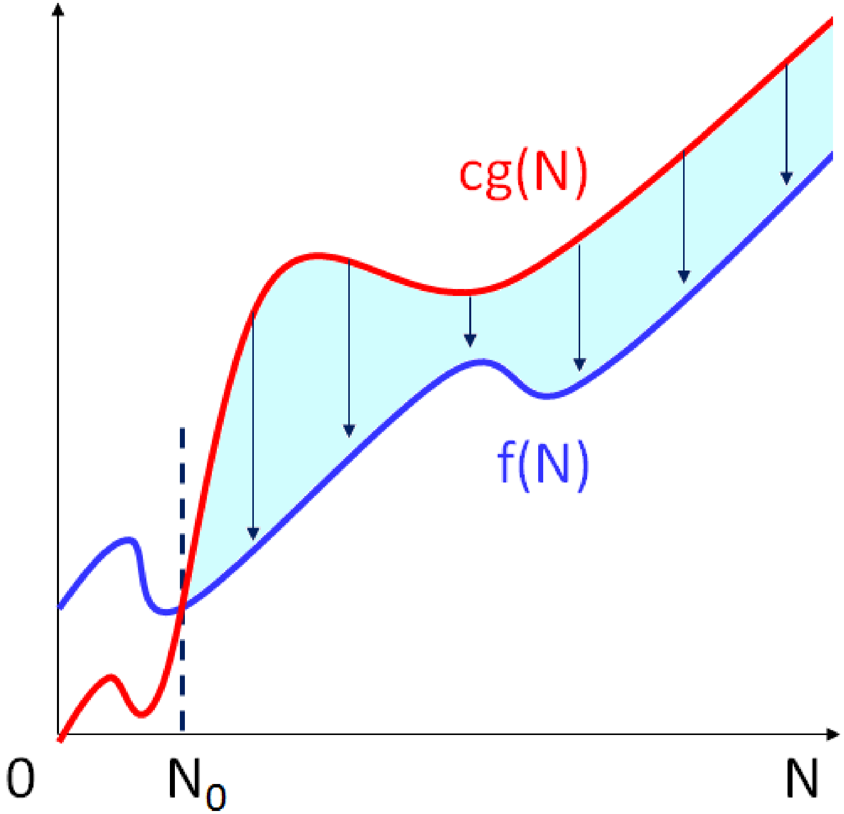
- 수행시간은 알고리즘이 수행하는 기본 연산 횟수를 입력 크기에 대한 함수로 표현
- 이러한 함수는 다항식으로 표현되며 이를 입력의 크기에 대한 함수로 표현하기 위해 점근표기법(Asymptotic Notation)이 사용
- O (Big-Oh)-표기법
- Ω (Big-Omega)-표기법
- Θ (Theta)-표기법

O (Big-Oh)-표기법

[O-표기의 정의]

- 두 개의 함수 f(n)과 g(n)이 주어졌을 때, 모든 $N \geq N_0$ 에 대해서 $f(N) \leq cg(N)$이 성립하는 양의 상수 c와N_0 이 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.



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

[예제]

- $f(N) = 2N^2 + 3N + 5$이면, 양의 상수 c 값을 최고 차항의 계수인 2보다 큰 4를 택하고 $g(N) = N^2$으로 정하면, 3보다 큰 모든 N에 대해 $2N^2 + 3N + 5 < 4N^2$이 성립, 즉, $f(N) = O(N^2)$
- 물론 $2N^2 + 3N + 5 = O(N^3)$도 성립하고, $2N^2 + 3N + 5 = O(2^N)$도 성립한다. 그러나 g(N)을 선택할 때에는 정의를 만족하는 가장 차수가 낮은 함수를 선택하는 것이 바람직함
- $f(N) \le cg(N)$을 만족하는 가장 작은 c값을 찾지 않아도 됨. 왜냐하면 $f(N) \le cg(N)$을 만족하는 양의 상수 c와 N_0가 존재하기만 하면 되기 때문

- 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.

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

- 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음
  - 예: T(n) = $n^2 + n + 1$
    - n=1일때 : T(n) = 1 + 1 + 1 = 3 (33.3%)
    - n=10일때 : T(n) = 100 + 10 + 1 = 111 (90%)
    - n=100일때 : T(n) = 10000 + 100 + 1 = 10101 (99%)
    - n=1,000일때 : T(n) = 1000000 + 1000 + 1 = 1001001 (99.9%)

n=100인 경우

$$T(n) = \underbrace{n^2}_{99\%} + \underbrace{n + 1}_{1\%}$$

Ω-표기법

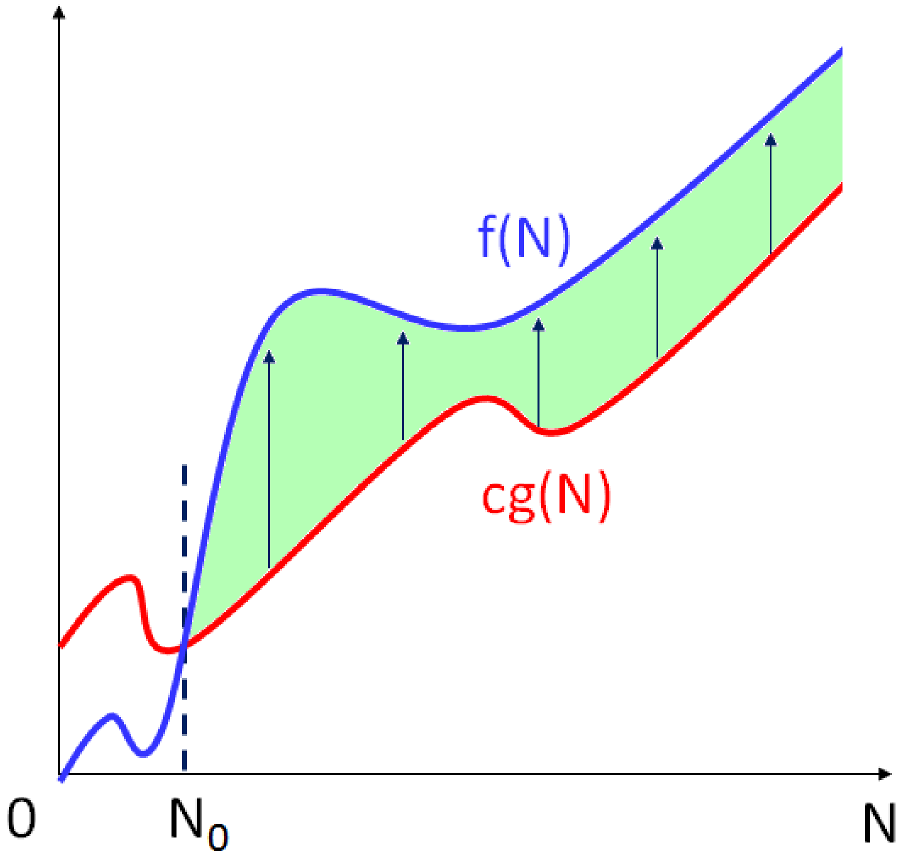
[Ω-표기의 정의]

- 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

[예제]

- $f(N) = 2N^2 + 3N + 5$ 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 N_0가 존재하기만 하면 되기 때문



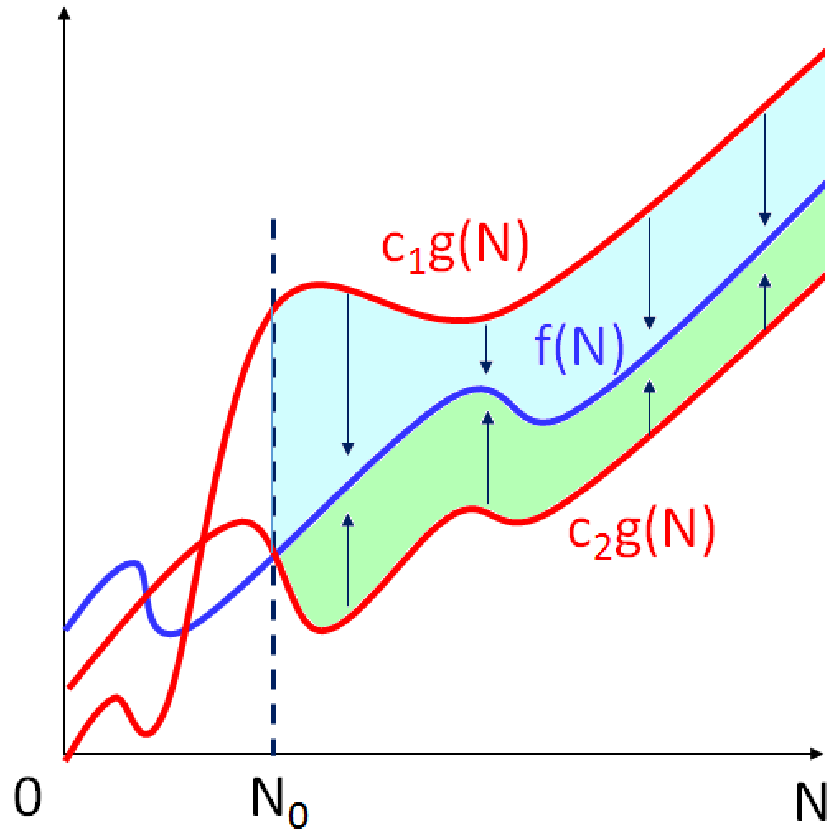
$$f(N) = \Omega(g(N))$$

Θ-표기법

[Θ-표기의 정의]

- 모든 $N \geq N_0$에 대해서 $c_1 g(N) \geq f(N) \geq c_2 g(N)$이 성립하는 양의 상수 c_1, c_2, 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 N^2과 $(2N^2 + 3N + 5)$이 유사한 증가율을 가지고 있다는 뜻
- $2N^2 + 3N + 5 \neq \Theta(N^3)$, $2N^2 + 3N + 5 \neq \Theta(N)$



$f(N) = \Theta(g(N))$

자주 사용되는 함수의 O-표기와 이름

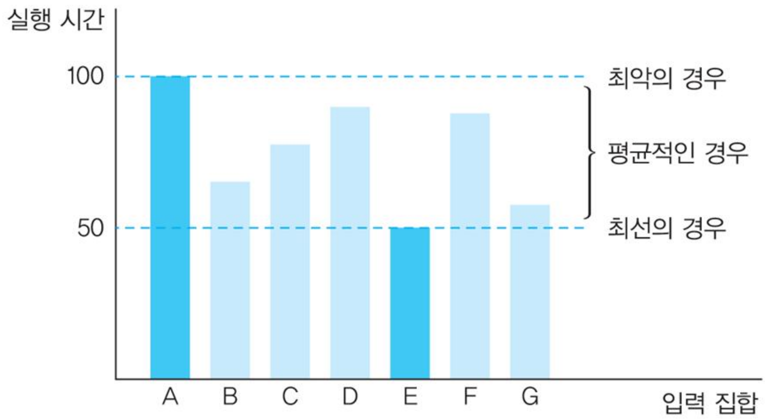
- 알고리즘의 수행시간은 주로 O-표기를 사용하며, 보다 정확히 표현하기 위해 Θ-표기를 사용하기도 한다.
- O(1) 상수시간(Constant Time)
- O(logN) 로그(대수)시간(Logarithmic Time)
- O(N) 선형시간(Linear Time)
- O(NlogN) 로그선형시간(Log-linear Time)
- $O(N^2)$ 제곱시간(Quadratic Time)
- $O(N^3)$ 세제곱시간(Cubic Time)
- $O(2^N)$ 지수시간(Exponential Time)

빅오 표기법의 종류

시간복잡도	n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40326	20922789888000	26313×10^{33}

최선, 평균, 최악의 경우

- 실행시간은 입력 집합에 따라 다를 수 있음
  - 최선의 경우(best case): 수행 시간이 가장 빠른 경우
    - 의미가 없는 경우가 많다.
  - 평균의 경우(average case): 수행시간이 평균적인 경우
    - 계산하기가 상당히 어려움.
  - **최악의 경우(worst case)**: 수행 시간이 가장 늦은 경우
    - 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.
      - (예) 비행기 관제업무, 게임, 로보틱스



등교 시간 분석

- 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

- 평균 시간: 대략 최악과 최선의 중간이라고 가정했을 때, 40분이 된다.

(a) 최선 경우

(b) 최악 경우

(c) 평균 경우

예 : 순차탐색의 최선, 평균, 최악

– 최선의 경우: O(1)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
5	**9**	**10**	**17**	**21**	**29**	**33**	**37**	**38**	**43**

인덱스 0에서 5 발견
숫자 비교 횟수 = 1

– 최악의 경우: O(n)

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
5	**9**	**10**	**17**	**21**	**29**	**33**	**37**	**38**	**43**

인덱스 0에서 43 발견
숫자 비교 횟수 = 10

– 평균적인 경우: $(1+2+\ldots+n)/n=(n+1)/2 \therefore O(n)$