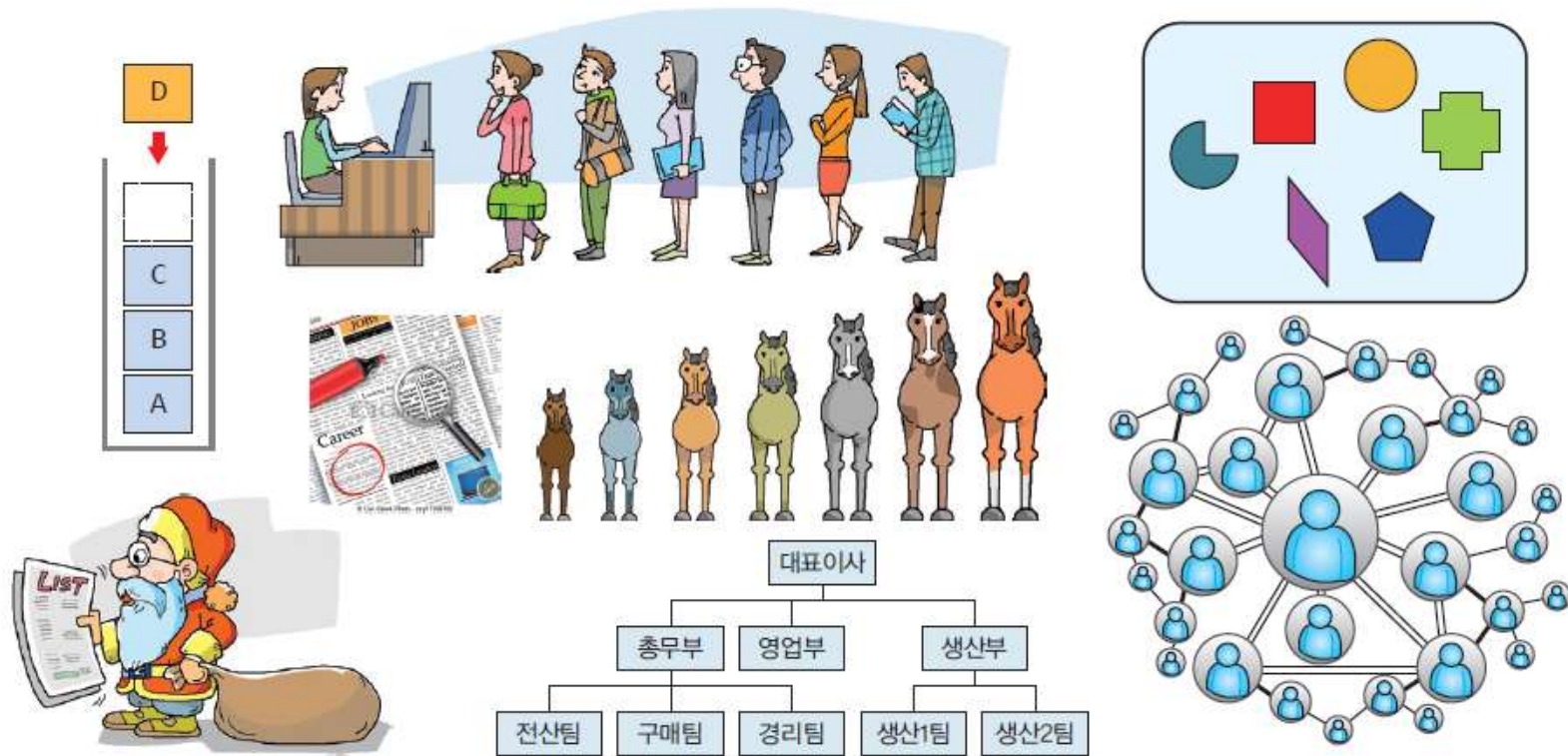


1주차 자료구조와 알고리즘

배 시 영

자료구조

- 일상 생활에서 사물의 조직화



자료구조

- 자료구조(Data Structure): 일련의 동일한 타입의 데이터를 정리하여 저장한 구성체
- 데이터를 정돈하는 목적: 프로그램에서 저장하는 데이터에 대해 탐색, 삽입, 삭제 등의 연산을 효율적으로 수행하기 위해서
- 자료구조를 설계할 때에는 데이터와 데이터에 관련된 연산들을 함께 고려해야 함.

자료구조

- 일상생활과 자료구조의 비교

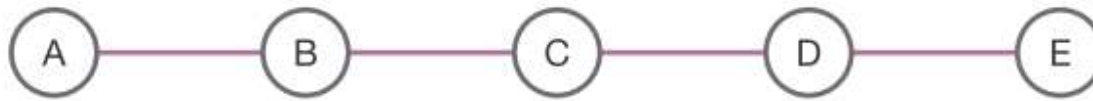
〈표 1-1〉 일상생활과 자료구조의 유사성

일상생활에서의 예	해당하는 자료구조
그릇을 쌓아서 보관하는 것	스택
마트 계산대의 줄	큐
버킷 리스트	리스트
영어사전	사전
지도	그래프
컴퓨터의 디렉토리 구조	트리

자료구조

■ 선형 자료구조

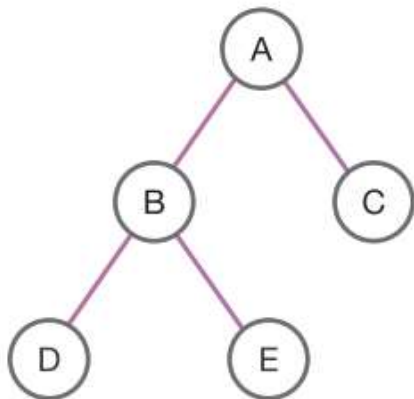
- 데이터를 한 줄로 순차적으로 표현한 형태. 선형 리스트, 연결 리스트, 스택, 큐 등



선형 자료구조의 형태

■ 비선형 자료구조

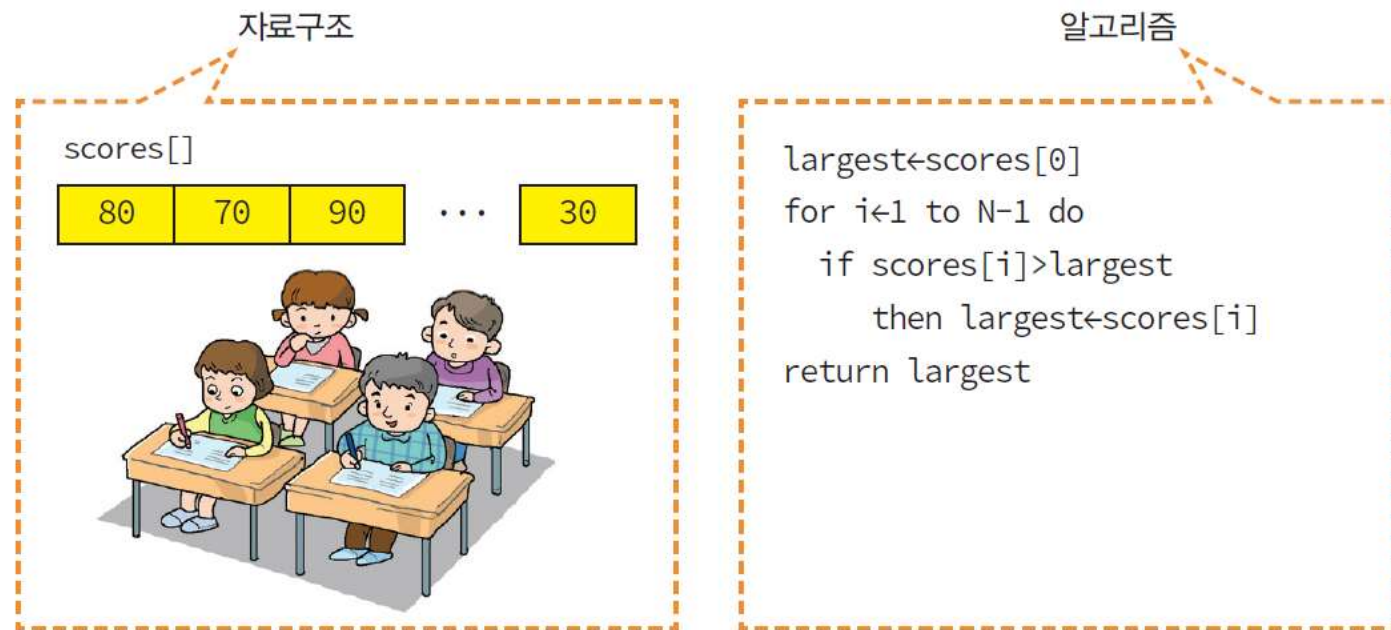
- 하나의 데이터 뒤에 여러 개가 이어지는 형태. 트리와 그래프 등



비선형 자료구조의 형태

알고리즘이란?

- 주어진 문제를 유한한 시간 내에 해결하는 단계적 절차
- 프로그램 = 자료구조 + 알고리즘



알고리즘 분석

- 관심사: “좋은” 알고리즘과 자료구조를 설계
- “좋은”의 척도
 - 알고리즘과 자료구조 작업에 소요되는 실행시간
 - 기억장소 사용량
- 어떤 알고리즘과 자료구조를 “좋다”고 분류하기 위해서는,
이를 분석하기 위한 정밀한 수단을 필요로 한다

실행시간의 분석

- 자료구조와 알고리즘의 효율성은 실행되는 연산의 수행시간으로 측정
- 자료구조에 대한 연산 수행시간 측정 방식은 알고리즘의 성능을 측정하는 방식과 동일
- 알고리즘의 성능: 실행시간을 나타내는 시간복잡도(Time Complexity)와 알고리즘이 실행되는 동안 사용되는 메모리 공간의 크기를 나타내는 공간복잡도(Space Complexity)에 기반하여 분석
- 대부분의 경우 시간복잡도만을 사용하여 알고리즘의 성능을 분석, 주어진 문제를 해결하기 위한 대부분의 알고리즘들이 비슷한 크기의 메모리 공간을 사용

시간복잡도

- 시간복잡도는 알고리즘(연산)이 실행되는 동안에 사용된 **기본적인 연산 횟수를 입력 크기의 함수로 나타낸다.**
- **기본 연산**(Elementary Operation)이란 탐색, 삽입이나 삭제와 같은 연산이 아닌, 데이터 간 크기 비교, 데이터 읽기 및 갱신, 숫자 계산 등과 같은 단순한 연산을 의미

실행시간

- 대부분의 알고리즘은 입력을 출력으로 변환한다
- 알고리즘의 **실행시간**(running time)은 대체로 **입력의 크기**(input size)와 함께 증가한다
- **평균실행시간**(average case running time)은 종종 결정하기 어렵다
- **최악실행시간**(worst case running time)에 집중
 - 분석이 비교적 용이
 - 게임, 재정, 로봇 등의 응용에서 결정적 요소

실행시간 구하기: 실험적 방법

- 알고리즘을 구현하는 프로그램을 작성
- 프로그램을 다양한 크기와 요소로 구성된 입력을 사용하여 실행
- 시스템콜을 사용하여 실제 실행시간을 정확히 측정
- 결과를 도표로 작성

실험의 한계

- 실험 결과는 **실험에 포함되지 않은 입력**에 대한 실행시간을 제대로 반영하지 않을 수도 있다
- 두 개의 알고리즘을 비교하기 위해서는, 반드시 **동일한 하드웨어와 소프트웨어 환경**이 사용되어야 한다
 - **HW**: processor, clock rate, memory, disk 등
 - **SW**: OS, programming language, compiler 등
- 알고리즘을 **완전한 프로그램으로 구현**해야 하는데, 이것이 매우 어려울 수가 있다

실행시간 구하기: 이론적 방법

- 모든 입력 가능성을 고려한다
- 하드웨어나 소프트웨어와 무관하게 알고리즘의 속도 평가 가능
 - 실행시간을 입력 크기, n 의 함수로 규정

알고리즘의 기술 방법

- 영어나 한국어와 같은 자연어
- 흐름도(flow chart)
- 의사 코드(pseudo-code)
- 프로그래밍 언어

자연어로 표기된 알고리즘

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

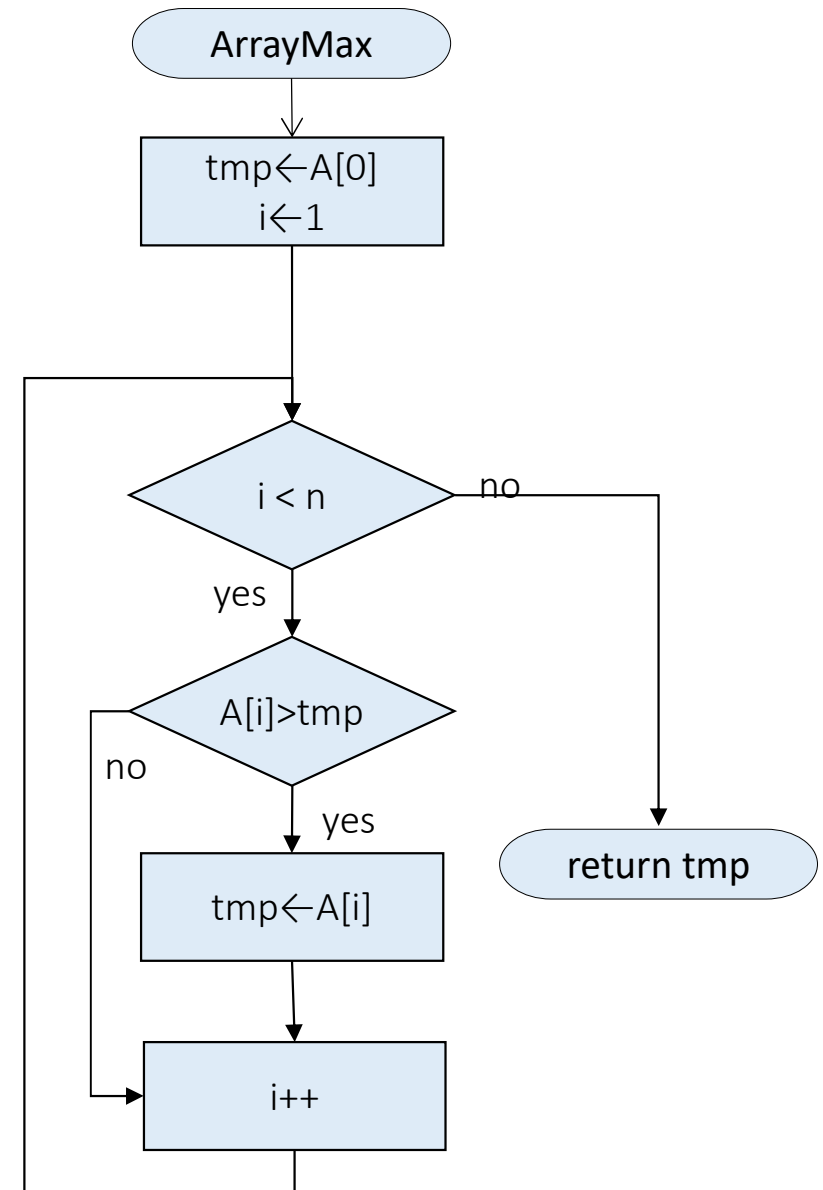
(예) 배열에서 최대값 찾기 알고리즘

ArrayMax(list, n)

1. 배열 list의 첫번째 요소를 변수 tmp에 복사
2. 배열 list의 다음 요소들을 차례대로 tmp와 비교하면 더 크면 tmp로 복사
3. 배열 list의 모든 요소를 비교했으면 tmp를 반환

흐름도로 표기된 알고리즘

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



특정언어로 표현된 알고리즘

- 알고리즘의 가장 정확한 기술이 가능
- 반면 실제 구현 시, 많은 구체적인 사항들이 알고리즘의 핵심적인 내용에 대한 이해를 방해할 수 있다.

```
def contains(bag, e) :           # bag에 항목 e가 있는지 검사하는 함수
    return e in bag             # 파이썬의 in 연산자 사용

def insert(bag, e) :            # bag에 항목 e를 넣는 함수
    bag.append(e)               # 파이썬 리스트의 append메소드 사용

def remove(bag, e) :            # bag에서 항목 e를 삭제하는 함수
    bag.remove(e)               # 파이썬 리스트의 remove메소드 사용

def count(bag):                 # bag의 전체 항목 수를 계산하는 함수
    return len(bag)             # 파이썬의 len 함수 사용
```

의사코드로 표기된 알고리즘

- **의사코드**(pseudo-code): 알고리즘을 설명하기 위한 고급언어
 - 컴퓨터가 아닌, 인간에게 읽히기 위해 작성됨
 - 저급의 상세 구현내용이 아닌, 고급 개념을 소통하기 위해 작성됨
- 자연어 문장보다 더 구조적이지만, 프로그래밍 언어보다 덜 상세함
- 알고리즘을 설명하는데 선호되는 표기법

◆ 예: 배열의 최대값 원소 찾기

```
Alg arrayMax(A, n)  
  input array A of n integers  
  output maximum element of A  
  
  1. currentMax  $\leftarrow A[0]$   
  2. for i  $\leftarrow 1$  to n - 1  
    if (A[i] > currentMax)  
      currentMax  $\leftarrow A[i]$   
  3. return currentMax
```

의사코드 문법

- 문법 이해 참고 사항

- exp : 수식
- var : 변수
- arg : 매개변수(인자)
- \dots : 임의의 명령문들
- $[x]$: x 가 선택적 구문 (x 가 없어도 상관 없음)
- $x \mid y \mid z$: x, y, z 택일
- x^* 는 x 가 0회 (없어도 상관 없음) 이상 반복 가능함

의사코드 문법

- 제어(control flow)

- if (*exp*) ...
[elseif (*exp*) ...]*
[else ...]

{0회 이상 중첩 elseif절 가능}
{else 절 생략 가능}

- for *var* ← *exp*₁ to|downto *exp*₂
...

{*var*의 값이 *exp*₁ 에서 *exp*₂ 이를 때까지 1씩
증가(혹은 감소) 하며 0회 이상 반복}

- for each *var* ∈ *exp*
...

{집합 *exp*이 포함하는 각 원소 *var*에 대해
0회 이상 반복}

- while (*exp*)
...

{*exp*이 참인 동안 0회 이상 반복}

- do
...
while (*exp*)

{*exp*이 참인 동안 1회 이상 반복}

- 주의: 들여쓰기(indentation)로 범위(scope)를 정의

의사코드 문법

- 연산(arithmetic)

- \leftarrow {치환(assignment)}
- $=, <, \leq, >, \geq$ {관계 연산자}
- $\&, \parallel, !$ {논리 연산자}
- $s_1 \leq n^2$ {첨자 등 수학적 표현 허용}

- 메소드(method) 정의, 반환, 호출

- Alg *method*([*arg* [, *arg*]*]) {메소드(알고리즘) 정의}
- ...
- return [*exp* [, *exp*]*] {메소드(알고리즘) 로 부터의 반환}
- *method*([*arg* [, *arg*]*]) {메소드(알고리즘) 호출}

- 주석(comments)

- input ... {메소드(알고리즘)의 입력 명세}
- output ... {메소드(알고리즘)의 출력 명세}
- {This is a comment} {코드 내 주석}

기본 연산

- 예
 - 산술식/논리식의 평가(EXP)
 - 변수에 특정값을 치환(ASS)
 - 배열원소 접근(IND)
 - 메소드 호출(CAL)
 - 메소드로 부터 반환(RET)

기본 연산 수 세기

- 의사코드를 조사함으로써, 알고리즘에 의해 실행되는 기본 연산의 최대 개수를 **입력크기**의 함수 형태로 결정할 수 있다

Alg *arrayMax*(*A*, *n*)

input array *A* of *n* integers

output maximum element of *A*

	{operations	count}
1. <i>currentMax</i> $\leftarrow A[0]$	{IND, ASS	2}
2. for <i>i</i> $\leftarrow 1$ to <i>n</i> - 1	{ASS, EXP	1 + <i>n</i> }
if (<i>A</i> [<i>i</i>] > <i>currentMax</i>)	{IND, EXP	2(<i>n</i> - 1)}
<i>currentMax</i> $\leftarrow A[i]$	{IND, ASS	2(<i>n</i> - 1)}
{increment counter <i>i</i> }	{EXP, ASS	2(<i>n</i> - 1)}
3. return <i>currentMax</i>	{RET	1}
	{Total	7 <i>n</i> - 2}

실행시간 추정

- *arrayMax*는 최악의 경우 $7n - 2$ 개의 기본 연산을 실행한다
- 다음과 같이 정의하자
 - a = 가장 빠른 기본 연산 실행에 걸리는 시간
 - b = 가장 느린 기본 연산 실행에 걸리는 시간
- 그리고 $T(n)$ 을 *arrayMax*의 최악인 경우의 시간이라 놓으면,
다음이 성립
$$a(7n - 2) \leq T(n) \leq b(7n - 2)$$
- 즉, 실행시간 $T(n)$ 은 두 개의 선형함수 사이에 놓이게 된다

실행시간의 증가율

- 하드웨어나 소프트웨어 환경을 변경하면:
 - $T(n)$ 에 상수 배수 만큼의 영향을 주지만,
 - $T(n)$ 의 증가율을 변경하지는 않는다
- 따라서 선형의 **증가율**(growth rate)을 나타내는 실행시간 $T(n)$ 은 *arrayMax*의 고유한 속성이다

실행시간의 점근표기법

- 실행시간은 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

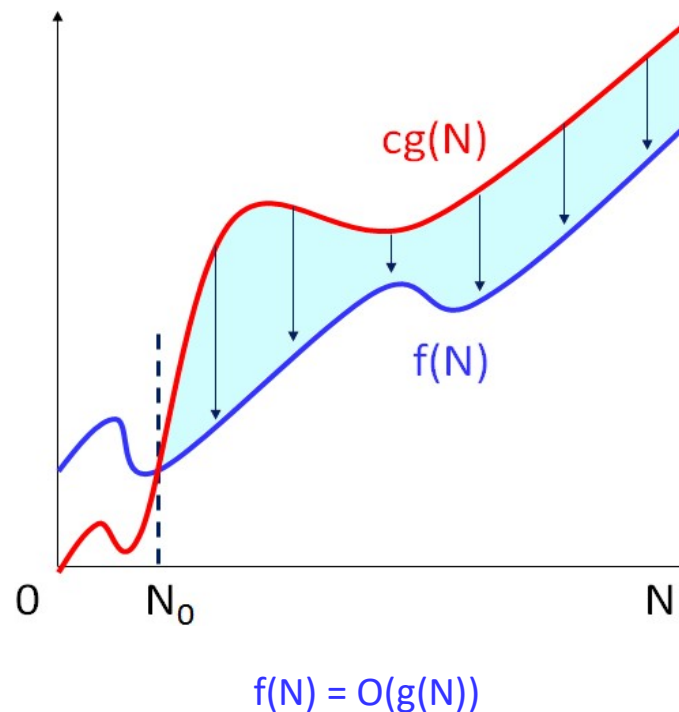
Big-Oh와 증가율

- Big-Oh 표기법은 함수의 증가율의 **상한**(upper bound)을 나타낸다
- “ $f(n) = O(g(n))$ ”이라 함은 “ $f(n)$ 의 증가율은 $g(n)$ 의 증가율을 넘지 않음”을 말한다
- Big-Oh 표기법을 사용함으로써, 증가율에 따라 함수들을 서열화할 수 있다

O (Big-Oh)-표기법

[O-표기의 정의]

- 주어진 두 개의 함수 $f(N)$ 과 $g(N)$ 에 관해, 만약 모든 $N \geq N_0$ 에 대해서 $f(N) \leq cg(N)$ 이 성립하는 양의 상수 c 와 N_0 이 존재하면, $f(N) = O(g(N))$ 이다.



O (Big-Oh)-표기법

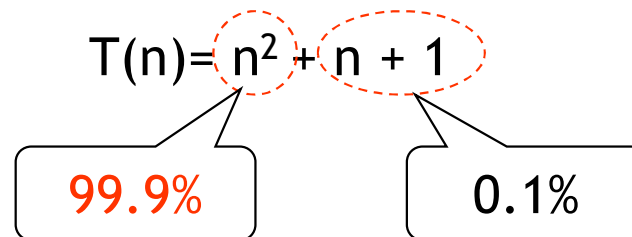
- 주어진 수행시간의 다항식에 대해 o-표기를 찾기 위해 간단한 방법은 **다항식에서 최고 차수 항만을 취한 뒤, 그 항의 계수를 제거**하여 $g(N)$ 을 정한다.
- 예를 들어, $2N^2 + 3N + 5$ 에서 최고 차수항은 $2N^2$ 이고, 여기서 계수인 2를 제거하면 N^2 이다.

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

O (Big-Oh)-표기법

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

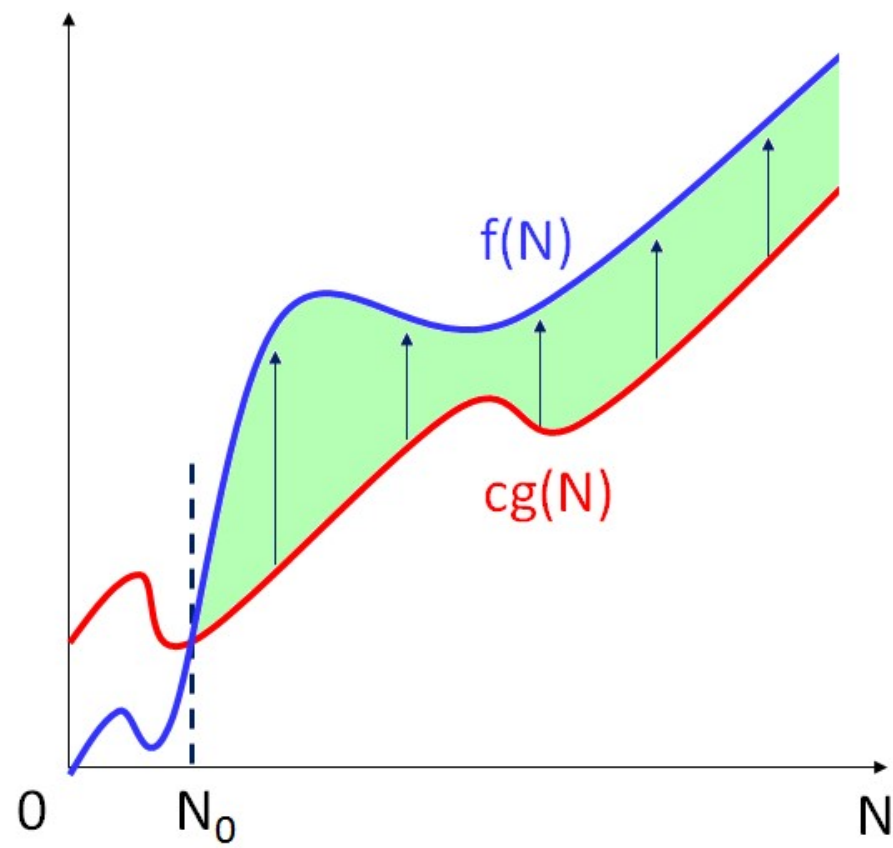
$n=1000$ 인 경우



Ω -표기법

[Ω -표기의 정의]

- 모든 $N \geq N_0$ 에 대해서 $f(N) \geq cg(N)$ 이 성립하는 양의 상수 c 와 N_0 이 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

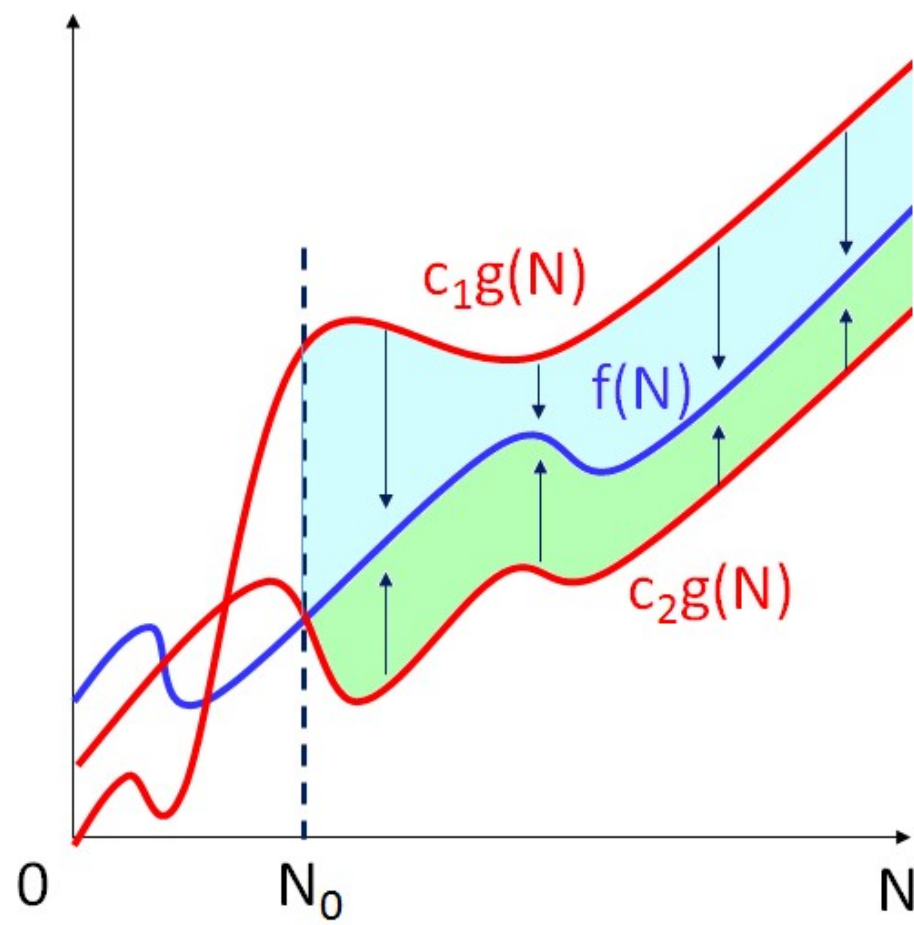


$$f(N) = \Omega(g(N))$$

Θ-표기법

[Θ-표기의 정의]

- 모든 $N \geq N_0$ 에 대해서 $c_1g(N) \geq f(N) \geq c_2g(N)$ 이 성립하는 양의 상수 c_1, c_2, N_0 가 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^2 과 $(2N^2 + 3N + 5)$ 이 유사한 증가율을 가지고 있다는 뜻



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

점근표기에 관한 직관

- Big-Oh
 - 점근적으로 $f(n) \leq g(n)$ 이면, " $f(n) = O(g(n))$ "
- Big-Omega
 - 점근적으로 $f(n) \geq g(n)$ 이면, " $f(n) = \Omega(g(n))$ "
- Big-Theta
 - 점근적으로 $f(n) = g(n)$ 이면, " $f(n) = \Theta(g(n))$ "

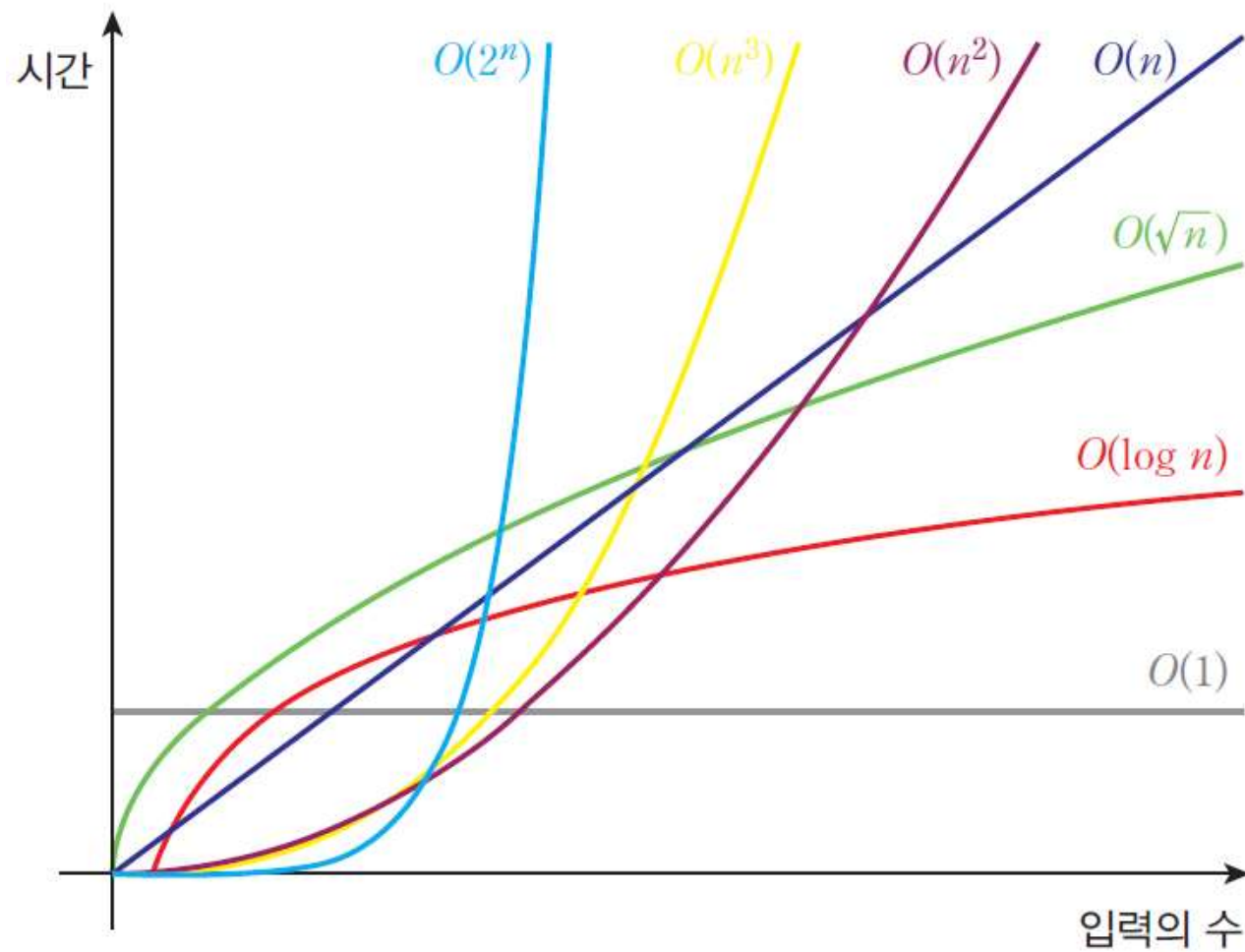
빅오 표기법의 종류

- $O(1)$ 상수시간(Constant Time)
- $O(\log N)$ 로그(대수)시간(Logarithmic Time)
- $O(N)$ 선형시간(Linear Time)
- $O(N \log N)$ 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
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	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

빅오 표기법의 종류



추상 자료형(ADT)

- 추상데이터타입(Abstract Data Type)은 **데이터와 그 데이터에 대한 추상적인 연산들**로써 구성
- ‘추상’의 의미: 연산을 구체적으로 어떻게 구현하여야 한다는 세부 명세를 포함하고 있지 않다는 의미
- 시스템의 핵심적인 구조나 동작에만 집중

자료형

- 자료형(data type): “데이터의 종류”
 - 정수, 실수, 문자열 등이 기초적인 자료형의 예
 - 데이터의 집합과 연산의 집합

int 자료형

데이터: $\{-\text{INT_MIN}, \dots, -2, -1, 0, 1, 2, \dots, \text{INT_MAX}\}$

연산: $+, -, *, /, \%, ==, >, <$

추상 데이터 타입의 정의

- **객체**: 추상 데이터 타입에 속하는 객체가 정의된다.
- **연산**: 이들 객체들 사이의 연산이 정의된다. 이 연산은 추상 데이터 타입과 외부를 연결하는 인터페이스의 역할을 한다.

예) 가방(Bag)의 추상 자료형

데이터: 중복된 항목을 허용하는 자료들의 저장소. 항목들은 특별한 순서가 없이 개별적으로 저장되지만 항목간의 비교는 가능해야 함.

연산:

- Bag(): 비어있는 가방을 새로 만든다.
- insert(e): 가방에 항목 e를 넣는다.
- remove(e): 가방에 e가 있는지 검사하여 있으면 이 항목을 꺼낸다.
- contains(e): e가 들어있으면 True를 없으면 False를 반환한다.
- count(): 가방에 들어 있는 항목들의 수를 반환한다.

예) 가방(Bag)의 추상 자료형의 구현

- 함수를 이용한 Bag 연산들의 구현 예(파이썬)

```
def contains(bag, e) :           # bag에 항목 e가 있는지 검사하는 함수
    return e in bag             # 파이썬의 in 연산자 사용

def insert(bag, e) :            # bag에 항목 e를 넣는 함수
    bag.append(e)               # 파이썬 리스트의 append메소드 사용

def remove(bag, e) :            # bag에서 항목 e를 삭제하는 함수
    bag.remove(e)              # 파이썬 리스트의 remove메소드 사용

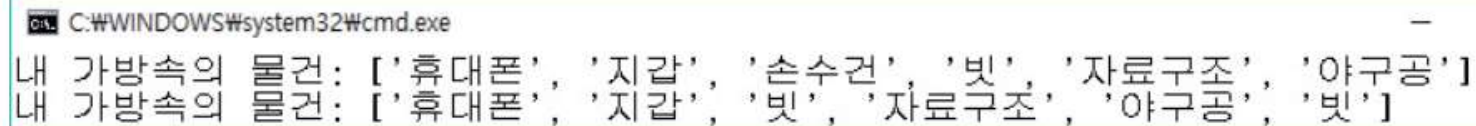
def count(bag):                 # bag의 전체 항목 수를 계산하는 함수
    return len(bag)            # 파이썬의 len 함수 사용
```

예) 가방(Bag)의 활용

- 가방(Bag)을 이용한 자료 관리 예

```
myBag = []                                # Bag을 위한 빈 리스트를 만들
insert(myBag, '휴대폰')                    # Bag에 휴대폰 삽입
insert(myBag, '지갑')                      # Bag에 지갑 삽입
insert(myBag, '손수건')                    # Bag에 손수건 삽입
insert(myBag, '빗')                       # Bag에 빗 삽입
insert(myBag, '자료구조')                  # Bag에 자료구조 삽입
insert(myBag, '야구공')                    # Bag에 야구공 삽입
print('가방속의 물건:', myBag)              # Bag의 내용 출력

insert(myBag, '빗')                        # Bag에서 '빗'삽입(중복)
remove(myBag, '손수건')                     # Bag에서 '손수건'삭제
print('가방속의 물건:', myBag)              # Bag의 내용 출력
```



```
C:\WINDOWS\system32\cmd.exe
내 가방속의 물건: ['휴대폰', '지갑', '손수건', '빗', '자료구조', '야구공']
내 가방속의 물건: ['휴대폰', '지갑', '빗', '자료구조', '야구공', '빗']
```



감사합니다!