

# 자료구조 & 알고리즘

for(A;B;C)  
D;

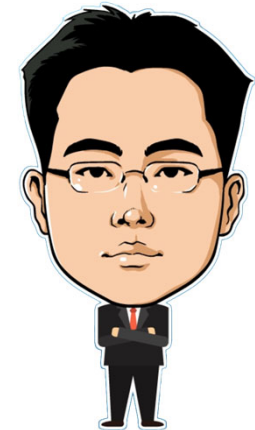


## 자료구조와 알고리즘 (Data Structures and Algorithms)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



# 목 차



- 자료구조와 알고리즘

- 알고리즘 설계와 분석



# 자료구조와 알고리즘



- 자료구조와 알고리즘

- 자료구조

- 알고리즘

- 알고리즘의 표현

- 알고리즘 설계와 분석



# 자료구조와 알고리즘

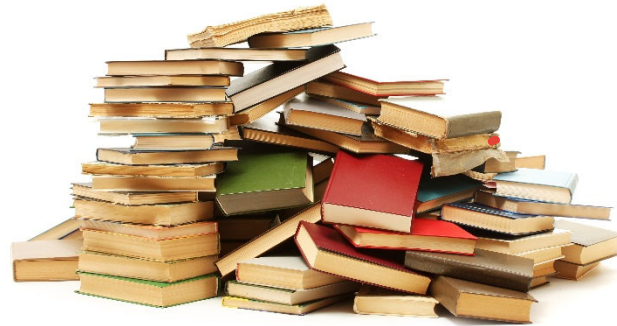
## 자료구조



# 자료구조 (1/6)

## ● 자료구조(Data Structures)

- 데이터를 저장, 조직, 관리하는 방법

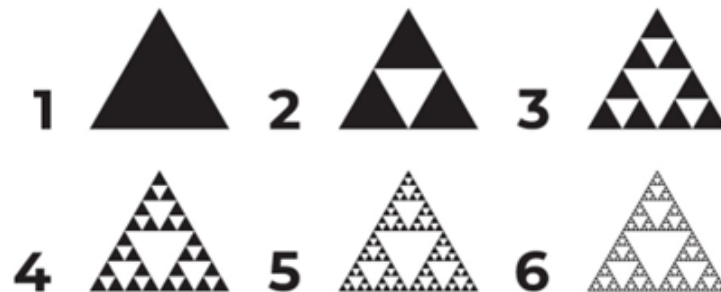


# 자료구조 (2/6)

## ● 자료구조

### ○ 생각하는 방법을 훈련하는 도구

- 자료구조는 그 자체로도 중요하다 못지 않게, 생각하는 방법 훈련도 중요하다.
  - 자료구조를 이용해서 문제를 해결하는 과정
  - 문제 해결 과정에서 논리의 골격이 구성되는 방법/스타일
  - 의미의 단위(의미의 매듭)를 설정하는 방법
- 재귀적 구조의 시에르핀스키 삼각형(Sierpinski Triangle)

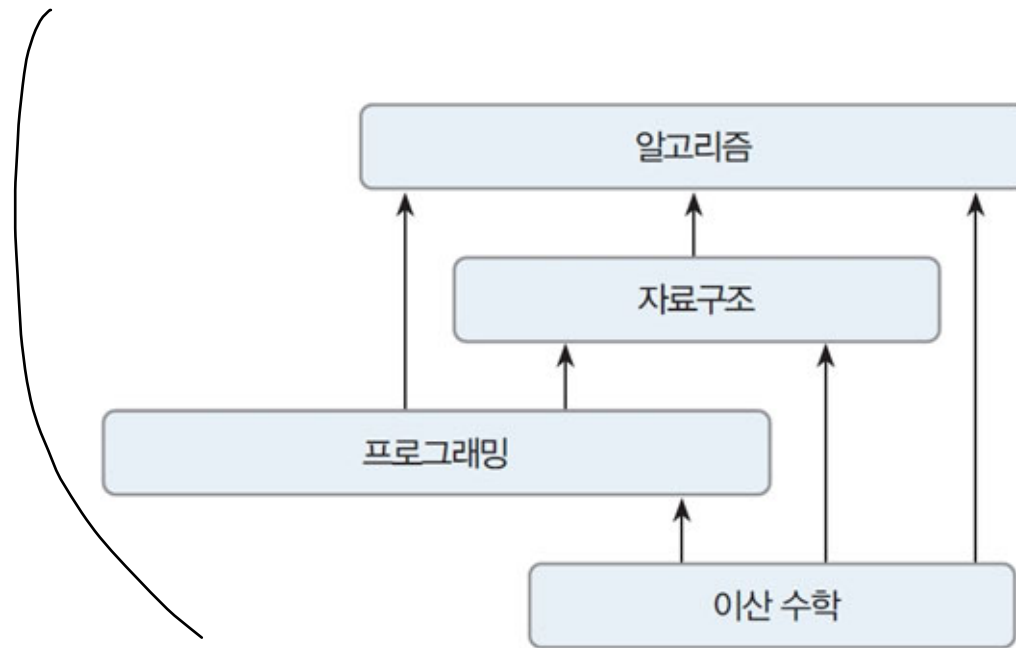


[ 이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022. ]

# 자료구조 (3/6)

- 자료구조

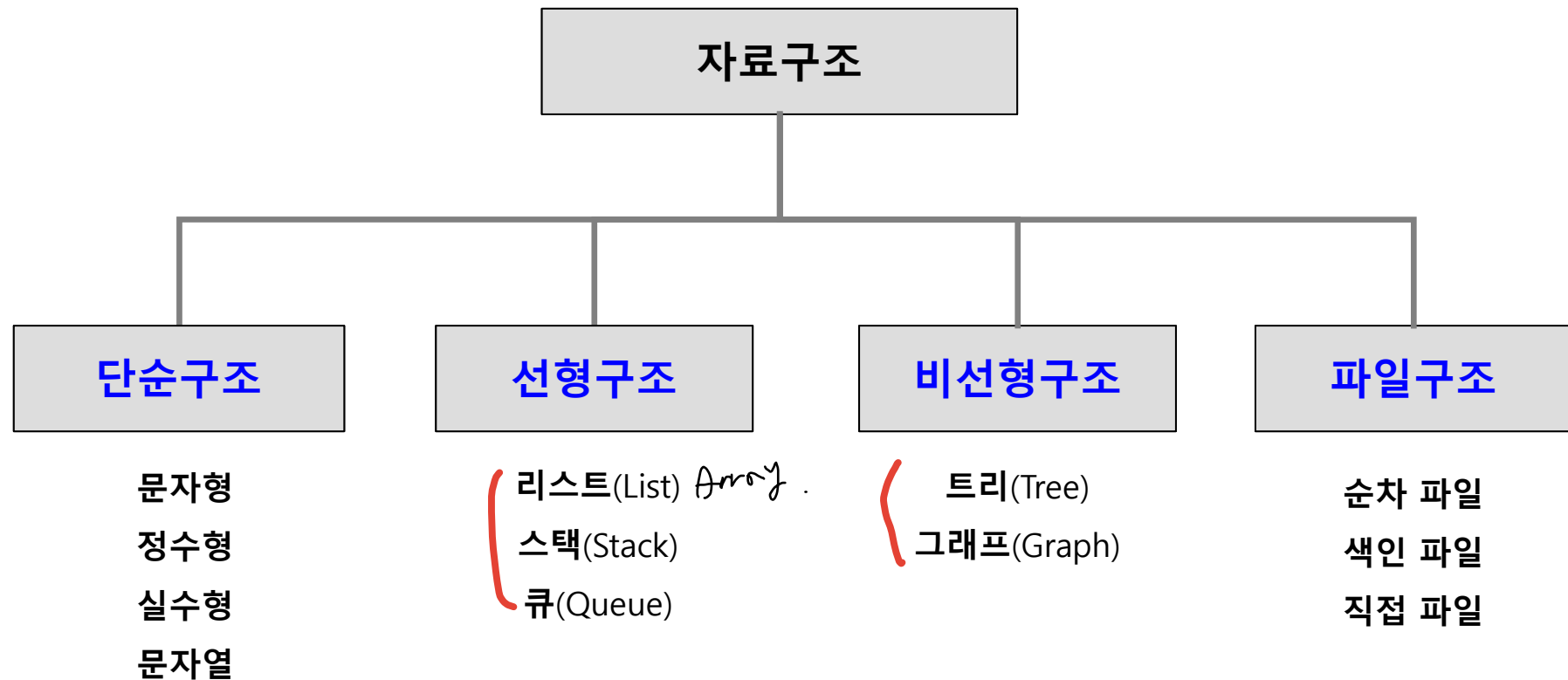
- 이산수학, 자료구조와 알고리즘, 프로그래밍의 관계



[ 이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022. ]

# 자료구조 (4/6)

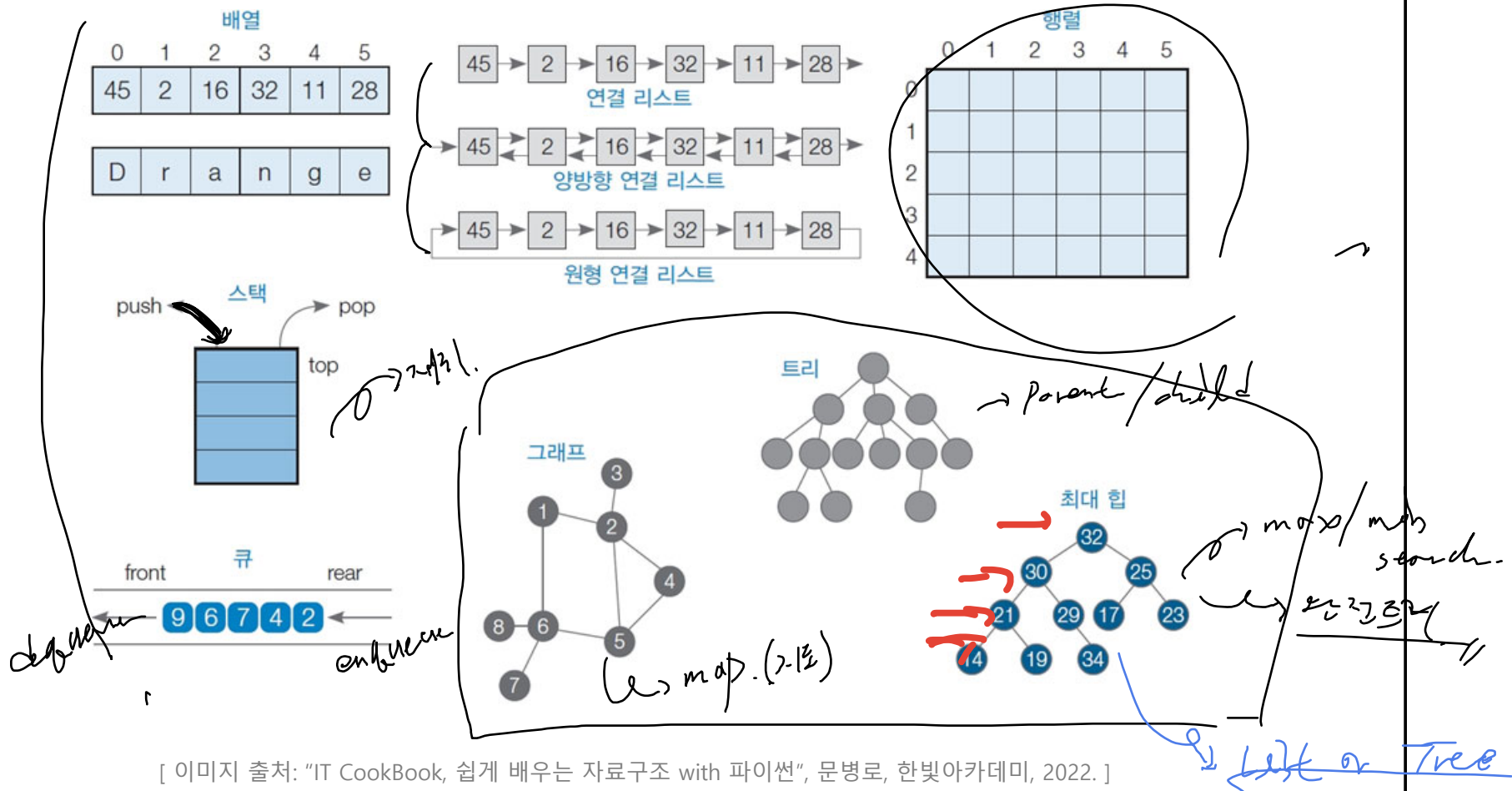
## ● 자료구조의 분류





# 자료구조 (5/6)

## ● 자료구조의 종류



[ 이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022. ]

# 자료구조 (6/6)

## ● 추상 데이터 타입 (ADT, Abstract Data Type)

### ○ 세부 사항에서 벗어나 추상적으로 정의한 데이터 타입

- 어떤 데이터 타입이 어떤 작업으로 이루어지는지만 표현한 것.
  - 내부에서 작업을 '어떻게' 하는지 신경 쓰지 않게 하며, '무엇을' 하는지만 신경 쓰게 해준다.
  - 함수, 클래스(Class)
- 추상화(Abstraction)
  - 중요하지 않은 부분을 무시하고, 중요한 부분에 집중함으로써 복잡성을 줄이는 기술
  - 파블로 피카소(Pablo Picasso), "예술은 필요 없는 것을 제거하는 과정이다."

# 자료구조와 알고리즘

## 알고리즘

생각하는 방법을 터득한 것은  
미래의 문제를 미리 해결한 것이다.

- 제임스 왓슨(James Dewey Watson)



# 알고리즘 (1/6)

## ● 알고리즘(Algorithm)

**“잘 정의된 문제 해결 과정”**

- 주어진 문제를 해결하기 위한 잘 정의된 동작들의 유한 집합
  - 어떤 작업을 수행하기 위해 입력을 받아 원하는 출력을 만들어내는 과정을 기술
  - 문제를 풀거나 작업을 수행하기 위한 단계적인 방법
- 자료구조와 알고리즘
  - 자료구조: 자료(행위의 객체: 무엇을)
  - 알고리즘: 문제 해결의 방법(행위적인 측면: 어떻게 하라)

**“무엇을 어떻게 하라”**



# 알고리즘 (2/6)

- 알고리즘: 알고리즘의 조건

- 알고리즘의 조건

1. 입력: 외부에서 제공하는 0개 이상의 입력이 존재
2. 출력: 1개 이상의 출력이 존재
3. 명백성: 각 명령어의 의미는 모호하지 않고 명확할 것
4. 유한성: 한정된 수의 명령어가 실행된 후에는 반드시 종료될 것
5. 유효성: 각 명령어들은 실행 가능한 연산일 것

알고리즘은 명확하고  
효율적으로 설계해야 한다.

# 알고리즘 (3/6)

- **수학적 알고리즘: 1부터 10까지의 합 구하기**

- 1부터 10까지의 합을 구하는 문제를 해결하는 세 가지 알고리즘

1. 1부터 10까지의 숫자를 직접 하나씩 더한다.

$$1 + 2 + 3 + \dots + 10 = 55$$

2. 두수의 합이 10이 되도록 숫자들을 그룹화하여, 그룹의 계수에 10을 곱하고 남은 숫자 5를 더한다.

$$(0 + 10) + (1 + 9) + (2 + 8) + (3 + 7) + (4 + 6) + 5 = 10 \times 5 + 5 = 55$$

3. 공식을 이용하여 계산할 수도 있다.

$$10 \times (1 + 10) / 2 = 55$$

# 알고리즘 (4/6)

## ● 알고리즘으로 어떤 문제를 푸는가?

### ○ 일상 생활에서 다양한 사례

- 아침에 일어 나서 옷 입는 순서
- 빵 만들기, 라면 조리법, 제품 설명서 등

### ○ 인터넷 검색: 빅 데이터 <sup>ex)</sup> Google

- 인터넷에는 수조 페이지 이상의 데이터가 존재한다.

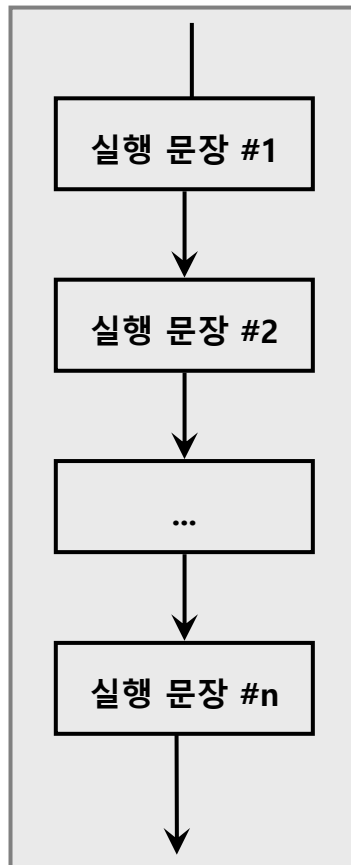
### ○ 자동차 내비게이션: 최단 경로 알고리즘

- 두 지점 간의 최단 경로나 최단 시간이 걸리는 경로 탐색
- 각 도로의 실시간 교통 상황을 제공하려면 더욱 복잡한 최단 경로 알고리즘 필요.

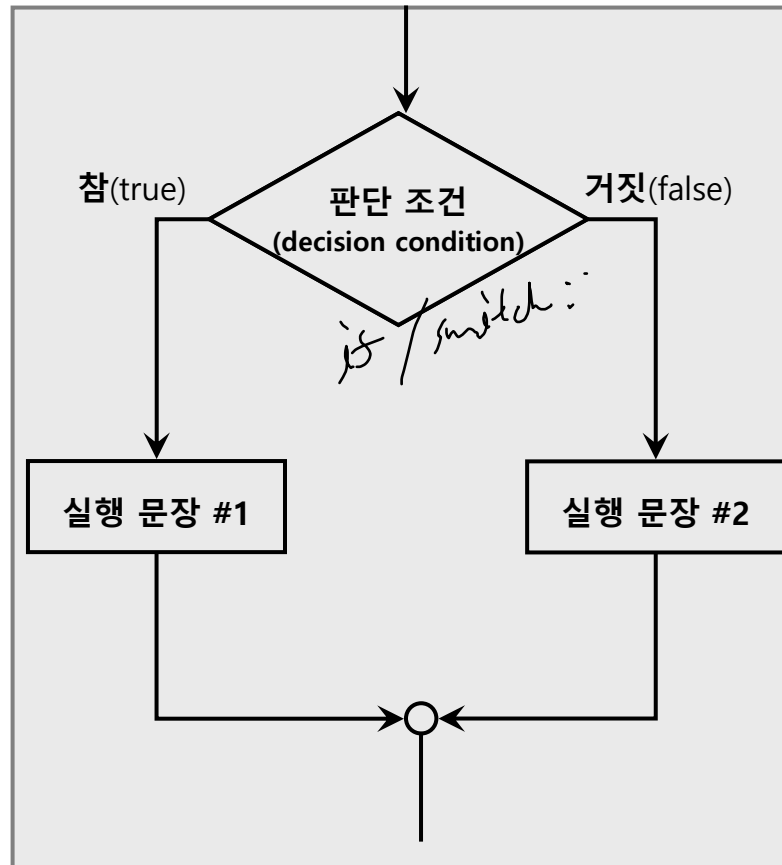
### ○ 신도시를 설계할 때 가스 파이프나 수도관은 어떻게 배치하는 것이 가장 효율적일까? : 최소 신장 트리.

# 알고리즘 (5/6)

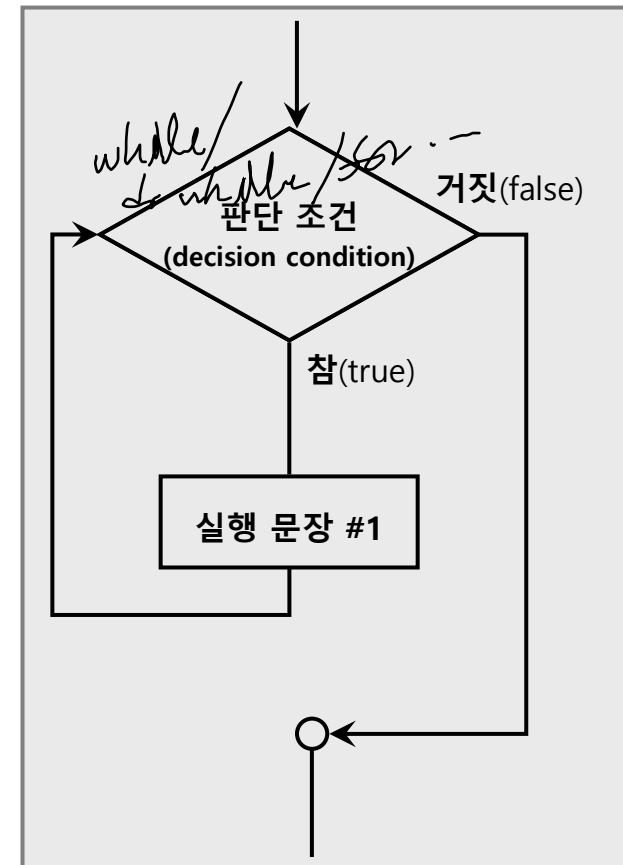
- **순서도**(flowchart): 알고리즘을 그림으로 표현



순차구조



선택구조

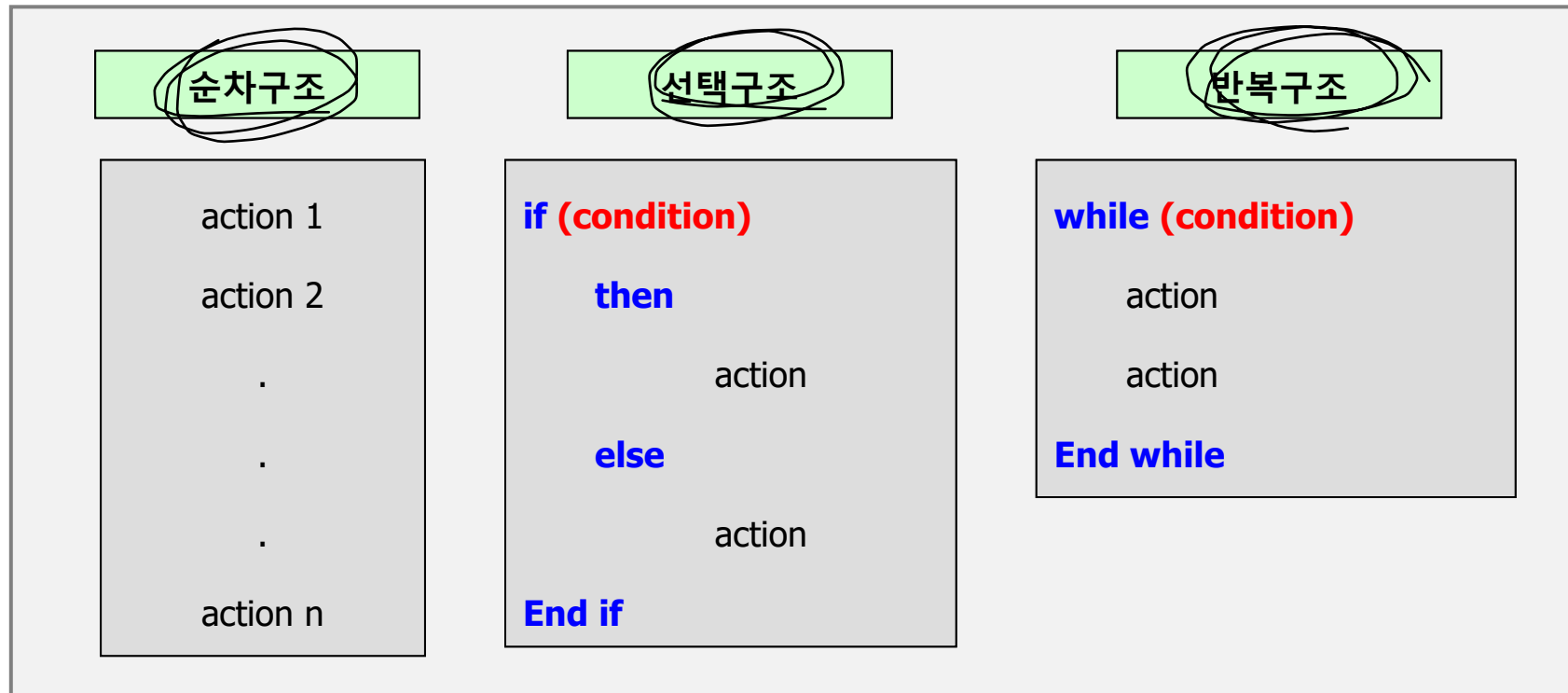


반복구조



# 알고리즘 (6/6)

- **의사코드(pseudo-code): 영어와 비슷한 자연어로 표현**
  - 특정 프로그래밍 언어의 문법에 따라 쓰인 것이 아니라, 일반적인 프로그래밍 언어와 형태가 유사하다.
    - 특정 언어로 프로그램을 작성하기 전에 알고리즘의 모델을 대략적으로 모델링



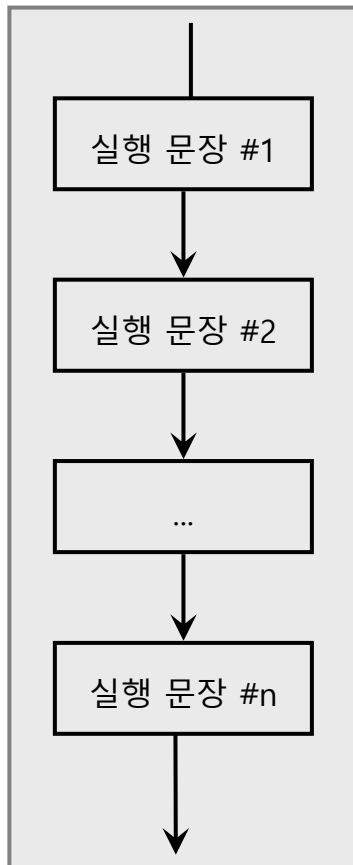
# 자료구조와 알고리즘

알고리즘의 표현: 순서도

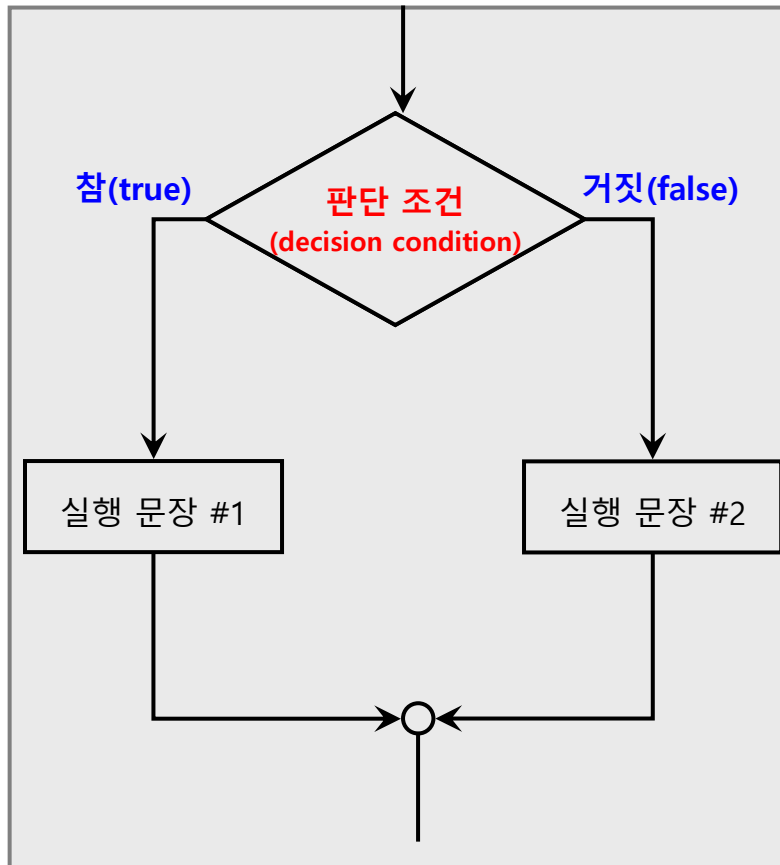


# 순서도 (1/7)

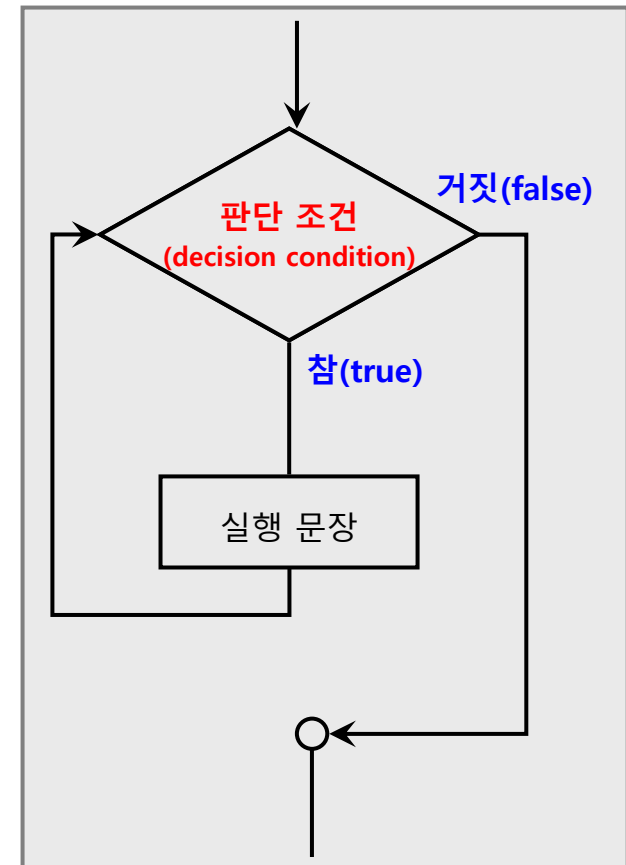
- **순서도**(Flowchart): 알고리즘을 그림으로 표현



순차구조



선택구조



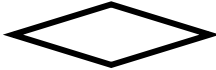



반복구조

# 순서도 (2/7)

- 순서도: 기호와 의미

- 순서도에 사용하는 기호와 의미

기 호	의 미	
	<u>단말</u> (Terminal)	순서도의 시작, 끝, 중단을 명시한다.
	<u>처리</u> (Process)	작동, 각종 연산, 값의 변화 등 처리해야 할 작업을 명시한다.
	<u>판단</u> (Decision)	주어진 조건(참, 거짓)에 따라 여러 흐름선 중 하나를 선택한다.
	키보드 입력(Input)	사용자가 키보드로 직접 입력한다.
	<u>입출력</u> (Input/Output)	데이터를 입력 또는 출력한다.

# 순서도 (3/7)

- **순서도: 기호와 의미**

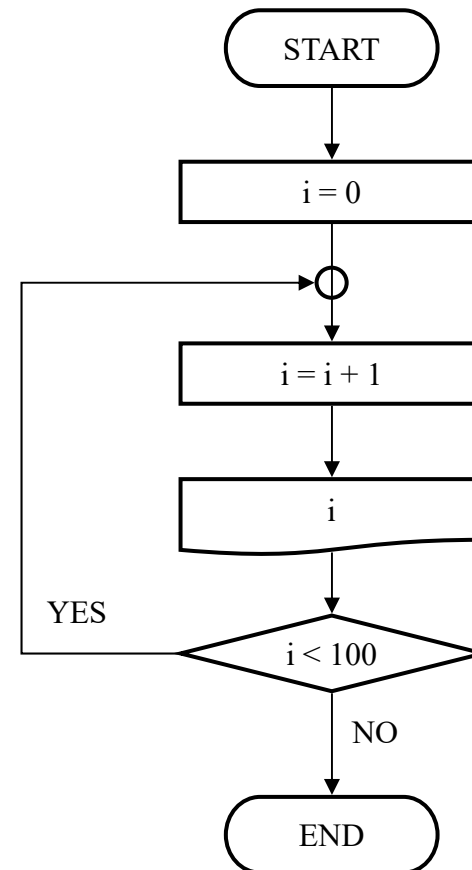
- 순서도에 사용하는 기호와 의미

기 호	의 미	
	서류(Document)	서류나 종이를 통한 입출력을 표시한다.
	<u>준비</u> (Preparation)	사전 준비과정(기억 장소의 할당, 초기값 설정)을 명시한다.
○	<u>연결</u> (Combination)	순서도의 다른 부분으로 연결한다.
	<u>흐름 방향</u> (Flow Direction)	처리의 흐름을 명시하고 기호를 연결한다.

# 순서도 (4/7)

- 알고리즘 예 #1

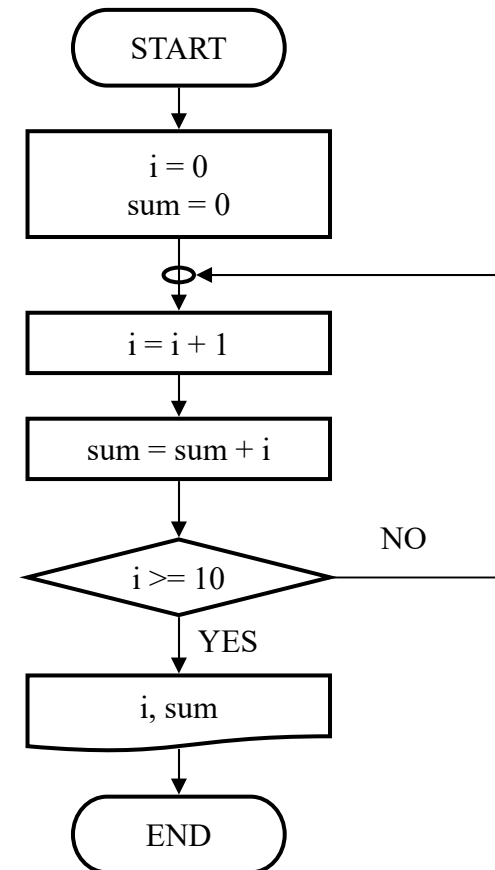
- 1에서 100까지 차례로 출력하기



# 순서도 (5/7)

## ● 알고리즘 예 #2

- 일정한 값으로 증가,  
감소하는 등차수열의 합
  - $1+2+3+\dots+10$ 과 같이 1부터 10까지 순서대로 증가하는 수열의 합계를 구하시오.



## 순서도 (6/7)

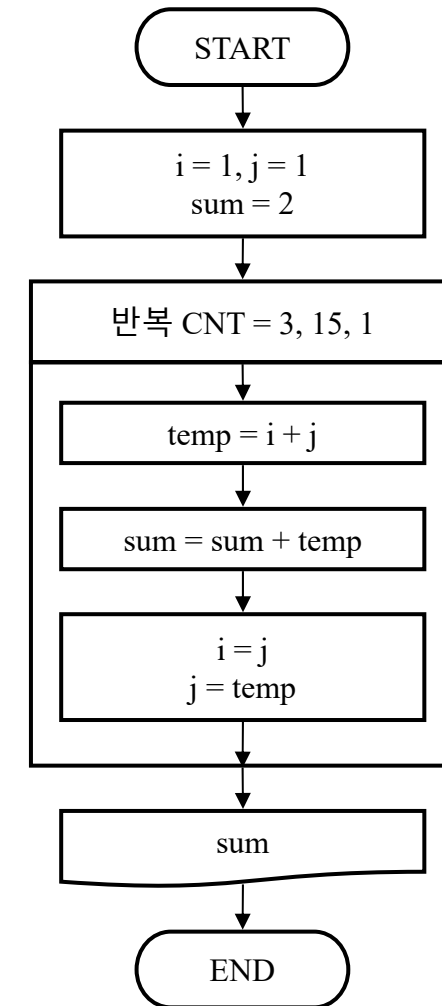
### ● 알고리즘 예 #3

#### ○ 피보나치 수열의 합계 구하기

- $1+1+2+3+5+8+13+\dots$ 와 같은 피보나치 수열의 15번째 항까지의 합계를 구하시오.

$$f_n = f_{n-1} + f_{n-2} (n \geq 3)$$

$$f_1 = f_2 = 1 (n = 1, 2)$$



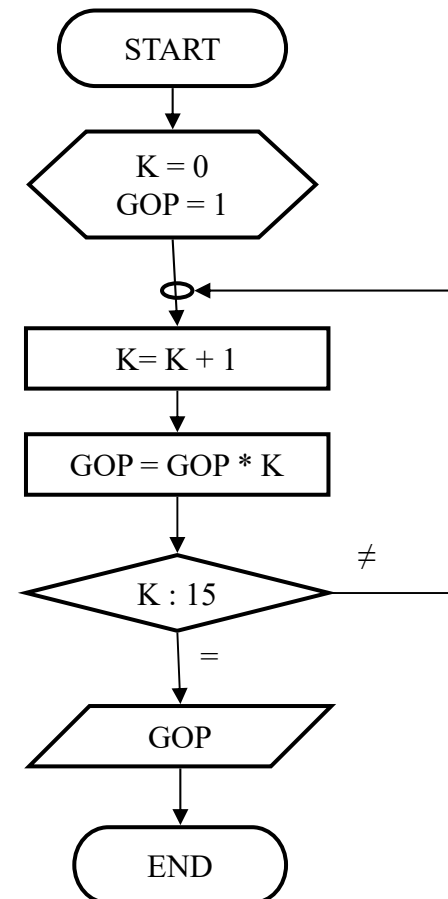


# 순서도 (7/7)

## ● 알고리즘 예 #4

### ○ 1에서 15까지의 누적 곱 (15! 구하기)

- $1 \times 2 \times 3 \times 4 \times 5 \times \dots \times 15$ 까지의 누적 곱 즉 15!을 구하시오.



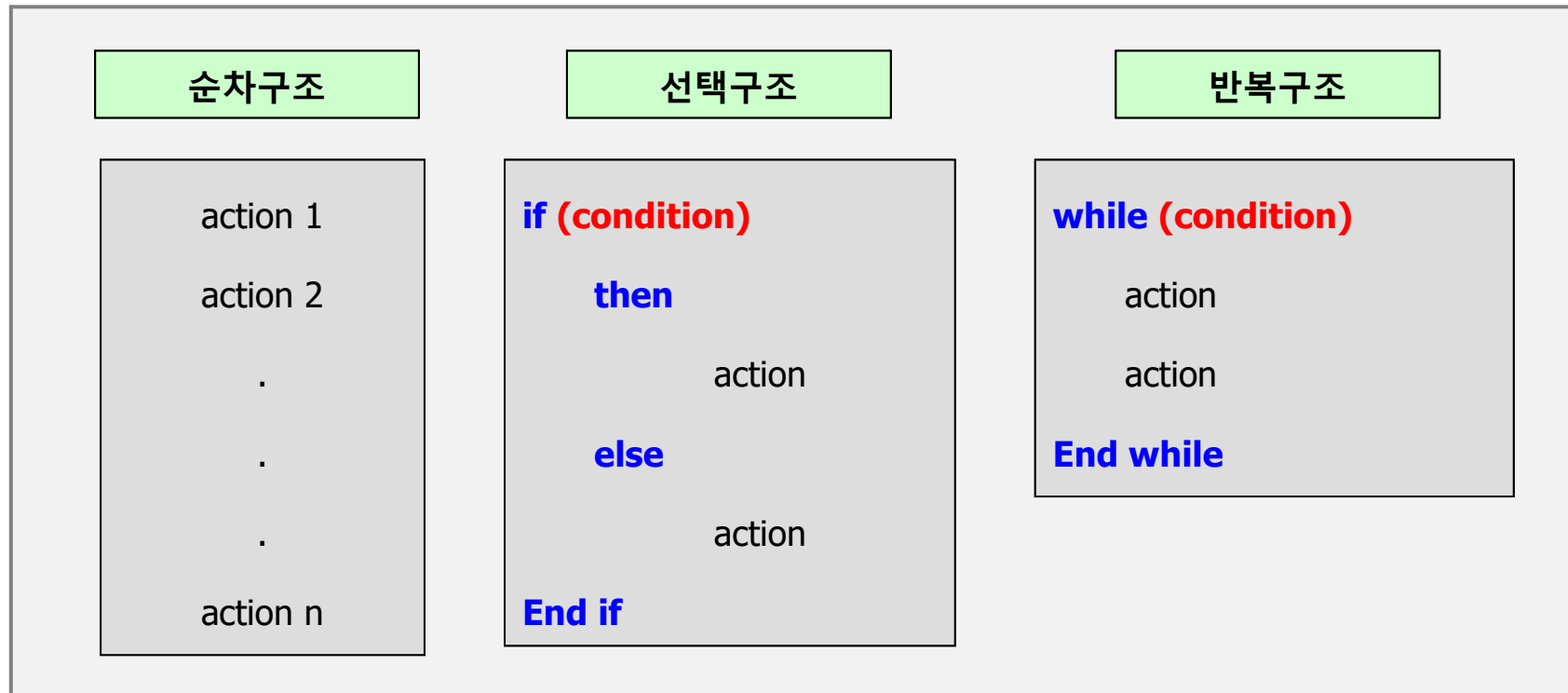
# 자료구조와 알고리즘

알고리즘의 표현: 의사코드



# 의사코드 (1/5)

- **의사코드**(pseudo-code): **영어와 비슷한 자연어로 표현**
  - 특정 프로그래밍 언어의 문법에 따라 쓰인 것이 아니라, 일반적인 프로그래밍 언어와 형태가 유사하다.
    - 특정 언어로 프로그램을 작성하기 전에 알고리즘의 모델을 대략적으로 모델링



# 의사 코드 (2/5)

- 알고리즘의 예: 두수의 평균

- 두 수의 평균을 구하는 알고리즘에 대한 알고리즘을 작성하라.

**AverageOfTwo** (평균구하기)

Input: 두 수

1. 두 수를 더한다.
2. 결과를 2로 나눈다.
3. 단계 2의 결과를 반환한다.

**End**

# 의사 코드 (3/5)

- 알고리즘의 예: 통과/비통과 성적

- 점수 성적을 통과/비통과 성적으로 변환 시키는 알고리즘을 작성하라.

Pass/NonPassGrade

Input: 한 개의 수

```
1. if (number가 70보다 크거나 같다.)  
    then  
        1.1 grade를 "pass"로 설정한다.  
    else  
        1.2 grade를 "nonpass"로 설정한다.  
    End if  
2. Grade를 반환한다.  
End
```

# 의사 코드 (4/5)

- 알고리즘의 예: 최대 수 찾기

- 정수의 집합에서 최대값을 찾는 알고리즘을 작성하라.
  - 단, 정수의 개수는 모른다.

## FindLargest

Input: 양의 정수 리스트

1. Largest를 0으로 설정한다.
2. while (다른 정수가 있다)
  - 2.1 if (정수가 Largest보다 크다)  
then  
2.1.1 Largest를 정수의 값으로 설정한다.  
End if
- End while
3. Largest를 반환한다.
- End

# 의사 코드 (5/5)

- 알고리즘의 예: 1000개의 숫자 중에서 최대 수 찾기

- 1000 개의 수 중에서 가장 큰 수를 찾는 알고리즘을 작성하라.

## FindLargest

Input: 1000개의 양의 정수

1. Largest를 0으로 설정한다.
2. Counter를 0으로 설정한다.
3. while (Counter가 1000보다 작다)
  - 3.1 if (정수가 Largest보다 크다)  
then
    - 3.1.1 Largest를 정수의 값으로 설정한다.
  - End if
  - 3.2 Counter를 증가 시킨다.
- End while
4. Largest를 반환한다.
- End

# 알고리즘 설계와 분석



- 자료구조
- 알고리즘 설계와 분석
  - 알고리즘 분석
  - 알고리즘의 수행 시간
  - 알고리즘 복잡도





# 알고리즘 설계와 분석

~~Speed & Memory~~

알고리즘 분석



# 알고리즘 분석 (1/2)

## ● 경험적 분석과 수학적 분석

### ○ 경험적 분석(Empirical analysis)

- 알고리즘을 프로그래밍 언어로 구현 후에 실행 시간을 비교해 보는 것
- 장점: 신빙성 있는 자료를 제공하며 수학적 지식이 필요 없다.

### ○ 수학적 분석(Mathematical analysis)

- 알고리즘 자체만을 가지고 수학적 분석을 하는 것
- 장점: 프로그래밍 과정에서 같은 알고리즘이더라도 프로그래머의 능력에 따라 나타날 수 있는 성능의 편차를 없앨 수 있다.

### ○ 최악의 경우와 최선의 경우

- ✓ 최악의 경우(worst case): 최악의 입력에 대한 분석 //
- 평균적 경우(average case): 모든 입력에 대한 분석(분석이 더 어렵다)
- 최선의 경우(best case): 최상의 입력에 대한 분석(별로 유용하지 않음)

“알고리즘의 성능을 나타내는 데에는  
최악의 경우에 한해서 나타내는 것이 보통이다.”

# 알고리즘 분석 (2/2)

- 알고리즘 분석의 단계

1. 알고리즘을 판단할 수 있는 입력 자료를 결정
2. 알고리즘을 구성하는 동작을 추상적이고 기본적인 동작들로 분해하여 그 동작들의 수행 시간을 계산.
3. 수학적인 알고리즘 분석

“알고리즘의 분석은 원하는 정도의 결과가 나올 때까지  
분석하고, 측정하고, 개선시키는 과정의 반복이다.”

# 알고리즘 설계와 분석

알고리즘의 수행 시간



# 알고리즘의 수행 시간 (1/8)

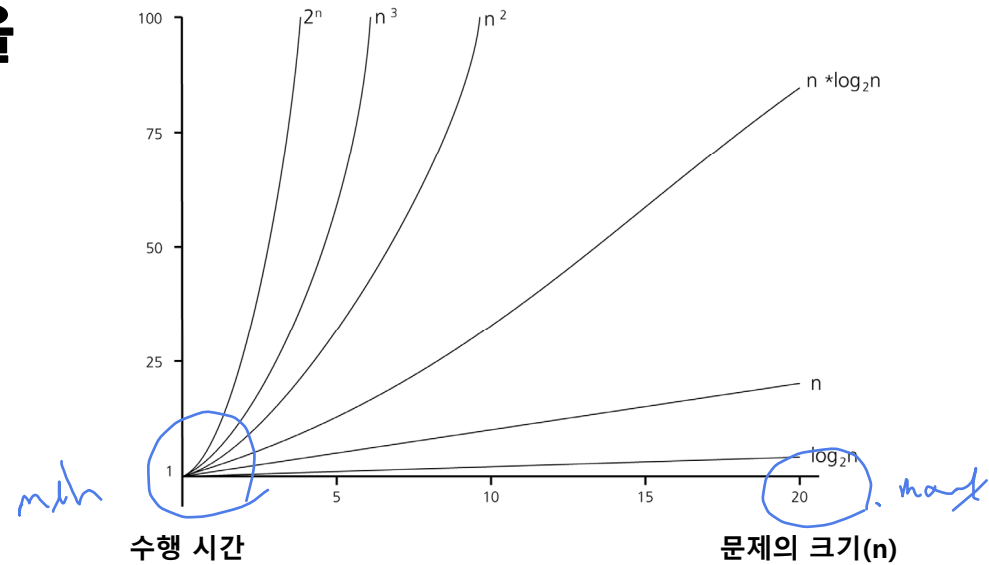
## ● 알고리즘의 수행 시간

### ○ 알고리즘의 효율성

- '자원을 얼마나 효율적으로 사용하는가'로 판단한다.
  - 자원: 시간, 저장 공간, 네트워크 대역 등
- 알고리즘의 수행 시간은 입력의 크기에 대해 시간이 얼마나 걸리는지로 표현한다.
  - 정렬의 경우: 정렬하고자 하는 개체(원소)의 수(크기)가 입력의 크기
  - 도시간의 최단 거리: 도시들의 총 수와 도시 간 간선(도로)의 총수가 입력의 크기
  - 팩토리얼(Factorial): 팩토리얼을 구하고자 하는 자연수의 크기가 입력의 크기
- 알고리즘의 수행 시간을 좌우하는 기준
  - for(또는 while) 문의 반복 횟수
  - 특정 행이 수행되는 횟수
  - 함수의 호출 횟수
  - 정렬의 경우: 주로 두 수를 비교하는 횟수 *sort is*

# 알고리즘의 수행 시간 (2/8)

- 여러가지 함수의 증가율



함수 \ n	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
$n$	10	$10^2$	$10^3$	$10^4$	$10^5$	$10^6$
$n * \log_2 n$	30	664	9,965	$10^5$	$10^6$	$10^7$
$n^2$	$10^2$	$10^4$	$10^6$	$10^8$	$10^{10}$	$10^{12}$
$n^3$	$10^3$	$10^6$	$10^9$	$10^{12}$	$10^{15}$	$10^{18}$
$2^n$	$10^3$	$10^{30}$	$10^{301}$	$10^{3,010}$	$10^{10,103}$	$10^{301,030}$

# 알고리즘의 수행 시간 (3/8)

## ● 알고리즘의 예 #1

- 입력의 크기가  $n$  : 간단한 연산

```
sample1(A[ ], n)
{
    k =  $\lfloor n/2 \rfloor$ 
    return A[k];
}
```

$O(1)$

$n$ 에 관계없이 상수 시간이 소요된다.

$\Omega(1)$  이기도 하고,  $O(1)$  이기도 하다.

$\Theta(1)$  : 가장 정확한 표현

*always*

“ $n$ 에 관계없이 상수 시간이 소요된다.”

# 알고리즘의 수행 시간 (4/8)

## ● 알고리즘의 예 #2

- 배열  $A[1, \dots, n]$  의 모든 원소를 더하는 알고리즘

```
sample2(A[ ], n)
```

```
{
```

```
    sum  $\leftarrow$  0 ;
```

```
    for i  $\leftarrow$  1 to n
```

```
        sum  $\leftarrow$  sum + A[i] ;
```

```
    return sum ;
```

```
}
```

$O(n)$   
ex) for

항상  $n$  에 비례하는 시간이 소요된다.

$\Omega(n)$  이기도 하고,  $O(n)$  이기도 하다.

$\Theta(n)$  가장 정확한 표현

always

- for 루프를 제외한 나머지 부분은 상수 시간이 소요되므로 for 루프가 시간을 지배한다.

“항상  $n$  에 비례하는 시간이 소요된다.”



# 알고리즘의 수행 시간 (5/8)

## ● 알고리즘의 예 #3

- 배열  $A[1, \dots, n]$  의 모든 원소 쌍을 곱한 합을 구하는 알고리즘

```
sample3(A[ ], n)
```

```
{
```

```
    sum ← 0 ;
```

```
    for i ← 1 to n
```

```
        for j ← 1 to n
```

```
            sum ← sum + A[i] * A[j] ;
```

```
    return sum ;
```

```
}
```

$O(n^2)$  ex) 곱셈 sum

항상  $n^2$  에 비례하는 시간이 소요된다.

$\Omega(n^2)$  이기도 하고,  $O(n^2)$  이기도 하다.

$\Theta(n^2)$ : 가장 정확한 표현

gross

- for 루프가 중첩되어 총  $n * n = n^2$  번 반복
- 각 루프에서는 덧셈 한 번과 곱셈 한 번(즉, 상수 시간 작업 수행)

**“항상  $n^2$  에 비례하는 시간이 소요된다.”**

# 알고리즘의 수행 시간 (6/8)

## ● 알고리즘의 예 #4

- 배열에서 반을 임의로 뽑아 그 중 최대값을 계속 더하는 알고리즘

```
sample4(A[ ], n)
{
    sum ← 0 ;
    for i ← 1 to n
    {
        for j ← 1 to n
        {
            k ← A[1 ... n] 에서 임의로  $\lfloor n/2 \rfloor$  개를 뽑을 때 이들 중 최대값 ;
            sum ← sum + k ;
        }
    }
    return sum ;
}
```

$O(n^3)$

n 에 비례하는 시간 소요

n

n

n

" $n^3$  에 비례하는 시간이 소요된다."

# 알고리즘의 수행 시간 (7/8)

## ● 알고리즘의 예 #5

- 배열  $A[1, \dots, n]$ 에서  $i < j$  인 모든 원소 쌍의 곱을 합산하는 알고리즘

```
sample5(A[ ], n)
```

```
{
```

```
    sum ← 0 ;
```

```
    for i ← 1 to n-1
```

```
        for j ← i+1 to n
```

```
            sum ← sum + A[i] * A[j] ;
```

```
    return sum ;
```

```
}
```

for 루프의 총 반복횟수:

$$(n-1) + (n-2) + \dots + 2 + 1 = \frac{n(n-1)}{2}$$

$O(n^2)$

" $n^2$ 에 비례하는 시간이 소요된다."

# 알고리즘의 수행 시간 (8/8)

- 알고리즘의 예 #6

- 재귀 호출 알고리즘

**factorial**(n)

```
{  
    if (n=1) then  
        return 1;  
    return n * factorial(n - 1);  
}
```

) →  $O(n)$   
⇒ (재귀 호출)  $O(n)$ .

"n에 비례하는 시간이 소요된다."

# 알고리즘 설계와 분석

## 알고리즘 복잡도

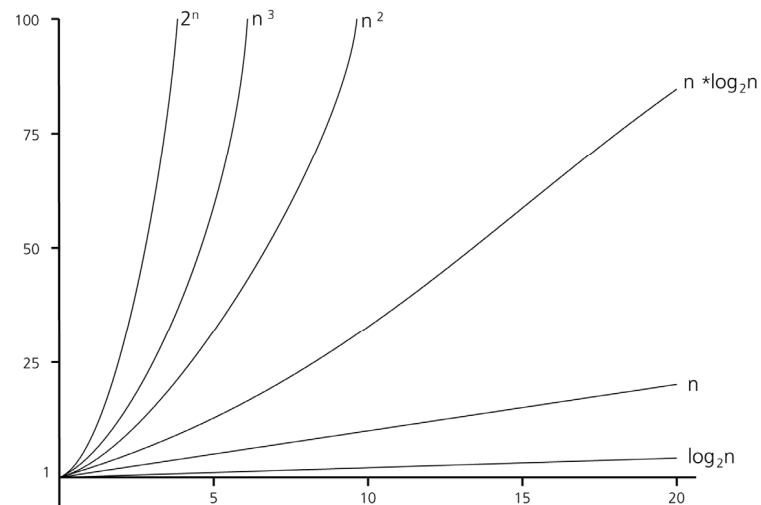


# 알고리즘 복잡도 (1/5)

- **점근적 분석** (Asymptotic Analysis)

- 알고리즘의 수행시간을 분석할 때는 항상 입력의 크기가 충분히 큰 때에 대해서 분석한다.

- 여러 함수의 증가율과 점근적 개념



$$\lim_{n \rightarrow \infty} f(n)$$

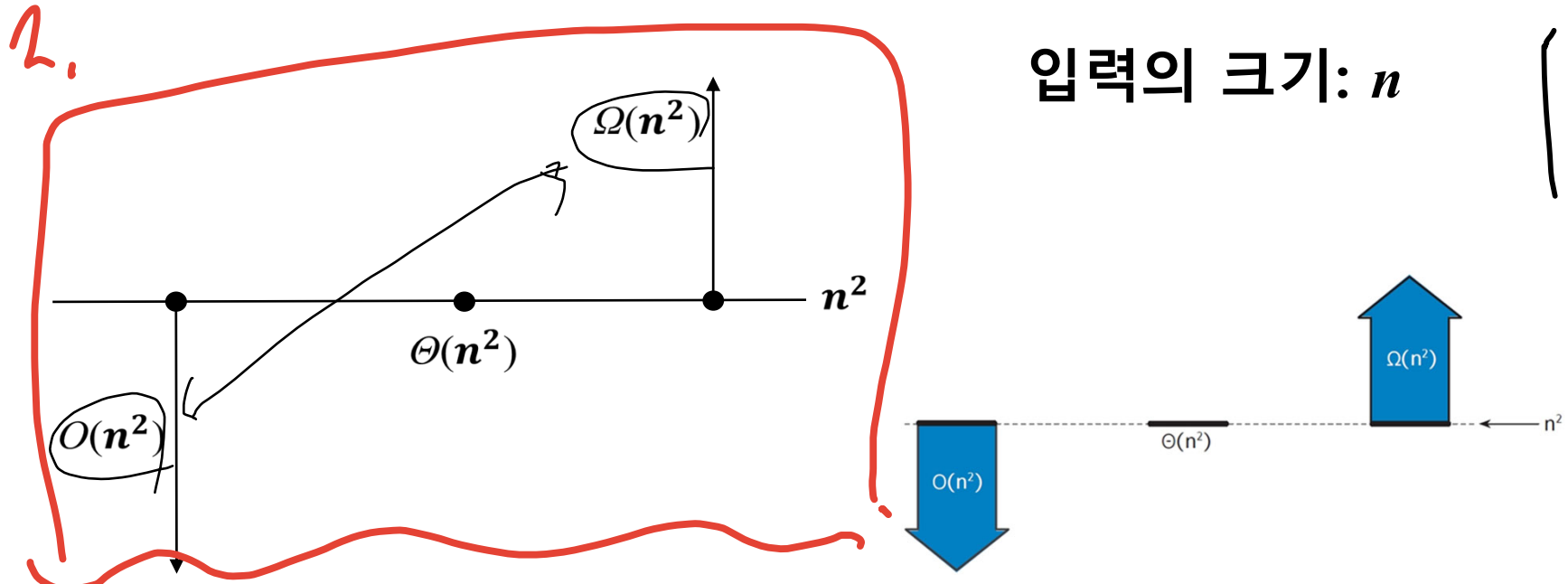
- 점근적 복잡도 (Asymptotic Complexity):  $O, \Omega, \Theta, o, \omega$  표기법

# 알고리즘 복잡도 (2/5)

## ● 점근적 분석: 점근적 복잡도

○  $O, \Omega, \Theta$  표기

- $O(n^2)$  알고리즘이 기껏해야  $n^2$  에 비례하는 시간이 든다.
- $\Omega(n^2)$  알고리즘이 적어도  $n^2$  에 비례하는 시간이 든다.
- $\Theta(n^2)$  알고리즘이 항상  $n^2$  에 비례하는 시간이 든다.



# 알고리즘 복잡도 (3/5)

## ● O : Big Oh 표기법

### ○ $O(f(n))$ : 점근적 상한

$\Rightarrow$  느려지지 않음

- 점근적 증가율이  $f(n)$ 을 넘지 않는 모든 함수들의 집합
- 최고차항의 차수가  $f(n)$ 과 일치하거나 더 작은 함수들의 집합
  - 예:  $O(n)$ ,  $O(n \log n)$ ,  $O(n^2)$ ,  $O(2^n)$ , ...
- $O(n^2)$  : 최고차항의 차수가  $n^2$  을 넘지 않는(이하) 모든 함수의 집합
  - $O(n^2) = \{n^2, 5n^2+7n+12, 100n^2-2n+9, n \log n+5n, 3n+9, 150, \dots\}$
- $5n^2+7n+12 \in O(n^2)$  으로 표기해야 하나 편의상  $5n^2+7n+12 = O(n^2)$  으로 표기함.

### ○ 직관적 의미

- $f(n) = O(g(n)) \Rightarrow f$ 는  $g$  보다 빠르게 증가하지 않는다.
- 상수 비율의 차이는 무시

### ○ O – 표기의 수학적 정의

- $O(g(n)) = \{f(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t. } \forall n \geq n_0, f(n) \leq cg(n)\}$
- $O(g(n)) = \{f(n) \mid \text{모든 } n \geq n_0 \text{ 에 대하여 } f(n) \leq cg(n) \text{인 양의 상수 } c \text{와 } n_0 \text{가 존재한다.}\}$
- $f(n) \in O(g(n))$ 을 관행적으로  $f(n) = O(g(n))$  이라고 쓴다.



# 알고리즘 복잡도 (4/5)

- $\Omega$  : Big Omega 표기법

- $\Omega(f(n))$  : 점근적 하한  $\Rightarrow$  적어도  $f(n)$ 의 비율로 증가하는 함수
  - 점근적 증가율이 적어도  $f(n)$ 의 비율로 증가하는 함수
  - 최고차항의 차수가  $f(n)$ 과 일치하거나 더 큰 함수들의 집합 ( $O(f(n))$  과 대칭적)
    - 예:  $\Omega(n)$ ,  $\Omega(n \log n)$ ,  $\Omega(n^2)$ ,  $\Omega(2^n)$ , ...
  - $\Omega(n^2)$  : 최고차항의 차수가  $n^2$  보다 작지 않은(이상) 모든 함수의 집합
    - $\Omega(n^2) = \{ n^2, 5n^2+7n+12, 100n^2-2n+9, n^3+5n, 7n^5+n^3+9, 2^n, n!, \dots \}$
  - $5n^3+12 \notin \Omega(n^2)$  으로 표기해야 하나 편의상  $5n^3+12 = \Omega(n^2)$  으로 표기함

- 직관적 의미

- $f(n) = \Omega(g(n)) \Rightarrow f$ 는  $g$  보다 느리게 증가하지 않는다.

- $\Omega$  - 표기의 수학적 정의

- $\Omega(g(n)) = \{ f(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t. } \forall n \geq n_0, cg(n) \leq f(n) \}$
- $\Omega(g(n)) = \{ f(n) \mid \text{모든 } n \geq n_0 \text{ 에 대하여 } cg(n) \leq f(n) \text{인 양의 상수 } c \text{와 } n_0 \text{가 존재한다.} \}$

# 알고리즘 복잡도 (5/5)

- 📌 : Big Theta 표기법

- $\Theta(f(n)) \Rightarrow$  동등

- 점근적 증가율이  $f(n)$ 과 일치하는 모든 함수들의 집합
- 최고차항의 차수가  $f(n)$ 과 일치하는 모든 함수들의 집합
  - 예:  $\Theta(n)$ ,  $\Theta(n \log n)$ ,  $\Theta(n^2)$ ,  $\Theta(2^n)$ , ...
- $\Theta(n^2)$ : 최고차항의 차수가  $n^2$  인 모든 함수의 집합
  - $\Theta(n^2) = \{n^2, 5n^2+7n+12, 100n^2-2n+9, \dots\}$
- $5n^2+12 \in \Theta(n^2)$  으로 표기해야 하나 편의상  $5n^2+12 = \Theta(n^2)$  으로 표기함

- 직관적 의미

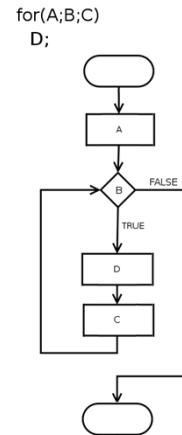
- $g(n) = \Theta(f(n)) \Rightarrow g$  는  $f$ 와 같은 정도로 증가한다.

- $\Theta$ -표기의 수학적 정의

- $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$

# 참고문헌

- [1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, "C++로 구현하는 자료구조와 알고리즘", 한티에듀, 2020.
- [2] 주우석, "IT CookBook, C · C++ 로 배우는 자료구조론", 한빛아카데미, 2019.
- [3] 이지영, "C 로 배우는 쉬운 자료구조", 한빛아카데미, 2022.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.
- [5] Richard E. Neapolitan, 도경구 역, "알고리즘 기초", 도서출판 홍릉, 2017.
- [6] "프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문", 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.
- [7] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [8] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,  
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

