



# 목차

**01**    컴퓨팅 사고의 이해

**02**    컴퓨팅 사고의 구성

**03**    알고리즘의 이해

**04**    알고리즘을 이용한 문제 해결

**05**    프로그래밍 언어

- 컴퓨팅 사고가 무엇인가를 이해한다.
- 컴퓨팅 사고의 구성요소인 추상화, 분해, 패턴인식, 알고리즘을 알아본다.
- 알고리즘의 표현 방식과 조건을 살펴본다.
- 프로그래밍 언어의 종류와 특징을 살펴본다.
- 컴파일러와 인터프리터의 차이점을 알아본다.

## ■ 일상생활에서 문제 해결

- 내비게이션의 최단 거리 검색
- 스마트 TV의 영화 추천

→ 인간이 만든 알고리즘을 컴퓨터에서 계산하여 그 결과를 보여 주는 것



그림 7-1 내비게이션을 활용한 최단 거리 찾기

## ■ 컴퓨팅 사고(Computational Thinking, CT)

- 문제를 해결하기 위해 논리적이고 창의적으로 생각하는 것
- 지넷 윙(Jeannette Wing) 박사가 기고한 논문에서 언급되었으며, 이후 많은 곳에서 많은 곳에서 컴퓨팅 사고의 개념을 도입



그림 7-2 컴퓨팅 사고를 처음 언급한 지넷 윙 박사

## ■ 컴퓨팅 사고의 정의

- 컴퓨터를 이용하여 문제를 해결하기 위한 논리적이고 창의적인 생각 방식



그림 7-3 컴퓨팅 사고

## ■ 버퍼와 공장 비유

- 사과를 잘게 부수는 것은 매우 빠르므로 기계에 사과를 하나씩 옮겨 집어넣는 것은 매우 비효율적
- 기계 속도를 맞추려면 사과를 큰 바구니에 담아 통째로 옮겨야 함
- 버퍼(큰 바구니) : 속도 차이가 많이 나는 두 장치 사이에 끼어서 속도 차이를 완화해 주는 장치



그림 7-6 사과주스 공장에서 버퍼 사용

## ■ 캐시와 조미료 통 비유

- 요리할 때마다 대용량 포장에서 조미료를 조금씩 덜어 쓰는 것은 매우 불편하므로 조미료를 조금씩 덜어 놓은 조미료 통 사용
- 캐시 : 앞으로 사용이 예상되는 것을 미리 가져다 놓은 것

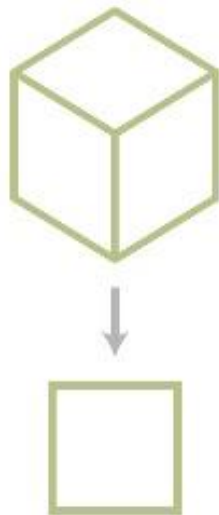


그림 7-7 캐시의 개념



## ■ 컴퓨팅 사고의 4가지 구성

- 추상화 : 문제에서 중요하지 않은 부분을 제거하고 중요한 특징만으로 문제를 구성함으로써 문제 해결을 좀 더 쉽게 하는 과정



(a) 추상화

## ■ 컴퓨팅 사고의 4가지 구성

- 분해 : 추상화한 문제를 해결하기 쉬운 작은 단위의 문제로 나누는 과정



(b) 분해

## ■ 컴퓨팅 사고의 4가지 구성

### ■ 패턴인식

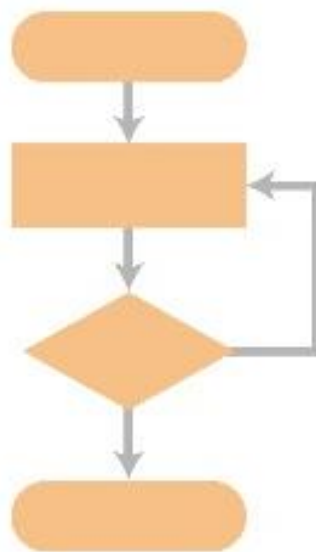
- 추상화 및 분해를 한 후, 데이터를 특징별로 나누어 유사한 문제 해결 방식이 있는지 찾아보는 과정
- 우리가 해결하는 문제 혹은 데이터에서 의미 있는 패턴을 찾아내는 과정



(c) 패턴 인식

## ■ 컴퓨팅 사고의 4가지 구성

- 알고리즘 : 어떠한 문제를 해결하기 위해 정해진 일련의 절차나 방법을 공식화한 형태로 표현한 것



(d) 알고리즘

## ■ 추상화의 개념

- 세부사항을 제거하여 간결하게 만드는 것
- 문제에서 불필요한 세부사항을 제거함으로써 문제 본질을 쉽게 파악할 수 있게 하는 작업

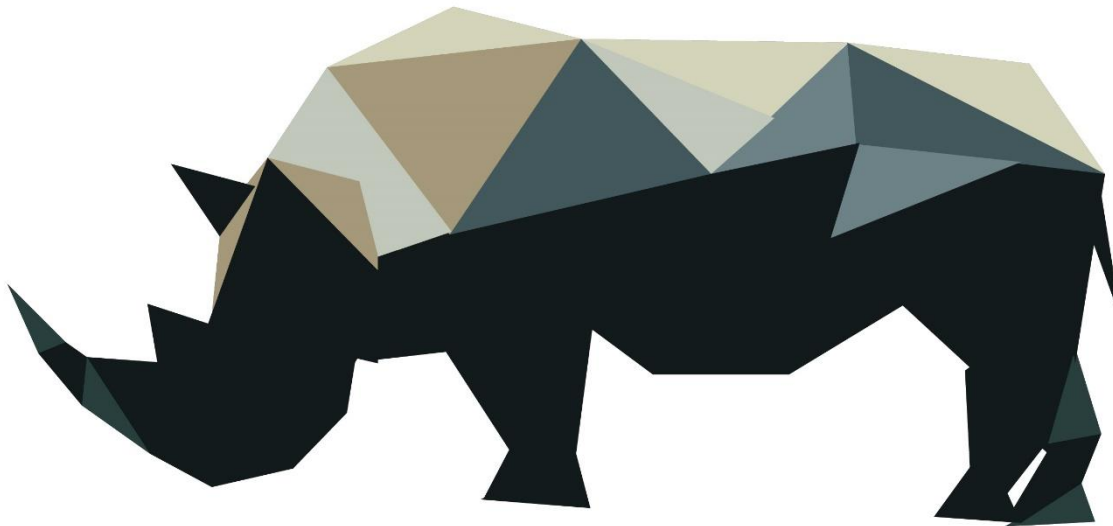


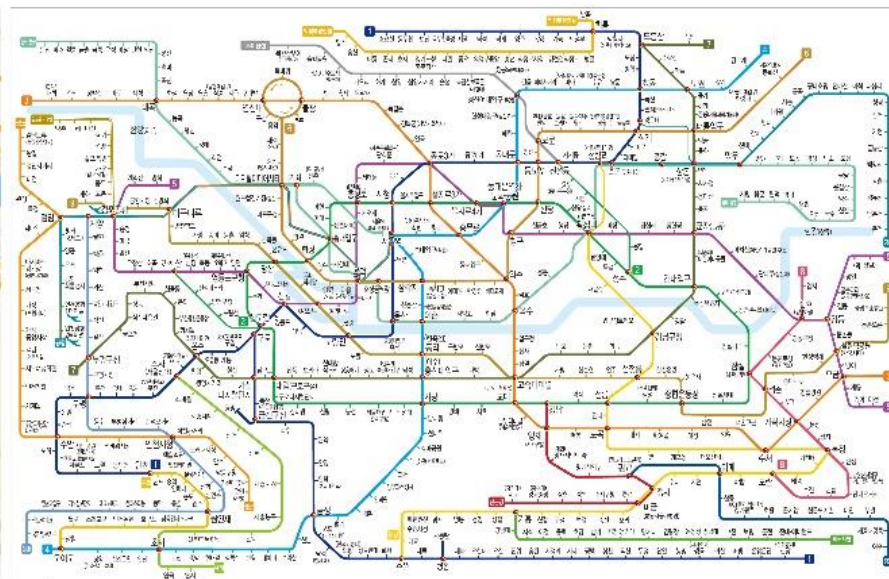
그림 7-10 다각형으로 추상화된 코뿔소

## ■ 추상화의 예

- 구글맵 : 한강뿐만 아니라 간선도로까지 모든 길이 다 표시되어 차로 이동할 때 편리
- 지하철 노선도 : 지하철로 이동할 때 구글맵은 필요 없는 정보가 너무 많아 지하철 정보만 있는 지하철 노선도가 편리



(a) 구글맵



(b) 지하철 노선도

그림 7-11 구글맵과 지하철 노선도

## ■ 추상화의 예

- 음식을 추상화해 놓은 네온사인(픽토그램)은 멀리 떨어진 곳에서도 무엇을 팔고 있는 가게인지 쉽게 인지
- 불필요한 것을 제거함으로써 문제 본질을 더욱 정확하게 파악할 수 있음



그림 7-12 음식 네온사인 (픽토그램)

## ■ 일반화

- 추상화로 공통의 특성(특징)을 추려 내어 만든 개념
- 수열의 특정 값  $X$ 는 바로 앞의 두 숫자를 더한 값으로  $X_n = X_{n-1} + X_{n-2}$ 처럼 공식으로 만들어 사용하면 이 수열의 특징을 나타낼 수 있음

$$1, 2, 3, 5, 8, 13, 21, 34, 55$$
$$\rightarrow X_n = X_{n-1} + X_{n-2}$$

**그림 7-13** 수열의 일반화



## ■ 문제 분해

- 복잡한 문제를 풀기 쉬운 간단한 문제로 나누는 것

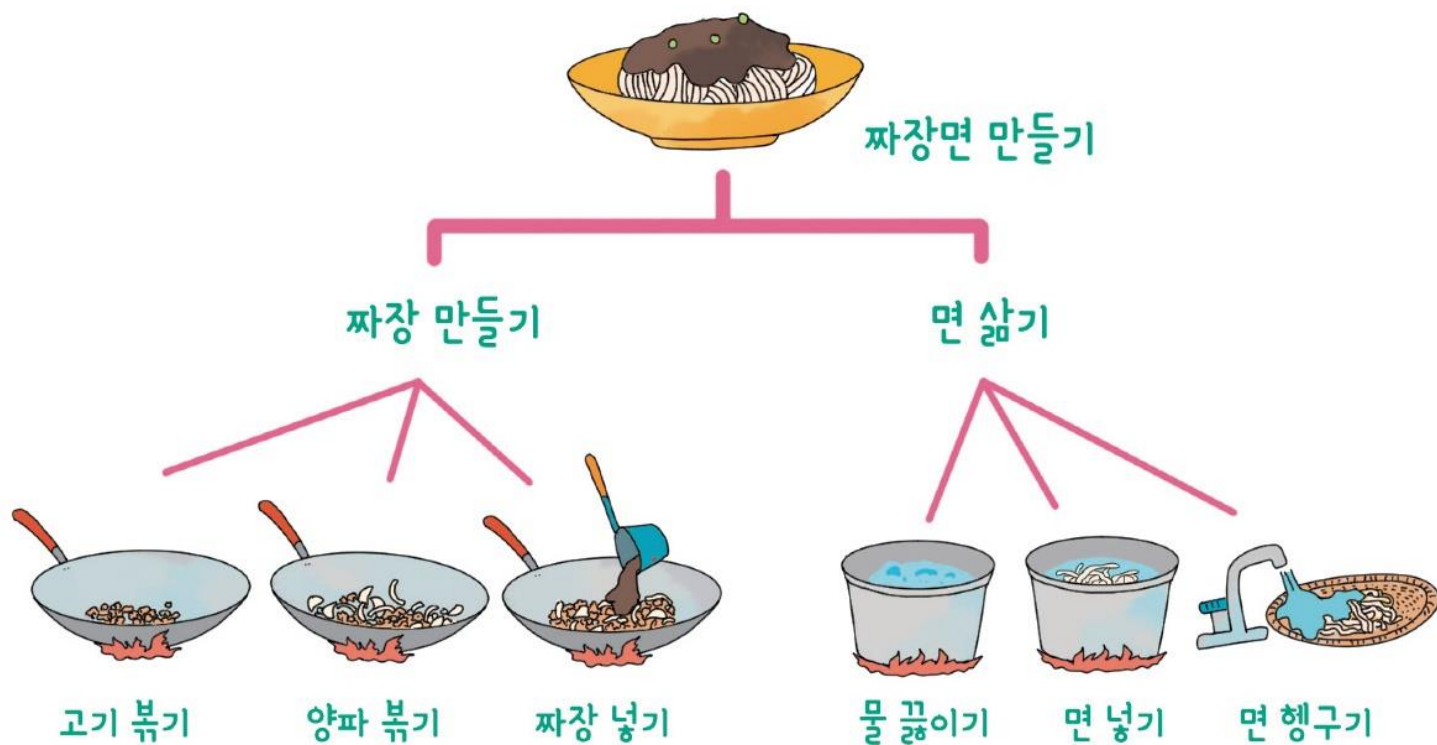


그림 7-14 짜장면 만들기의 문제 분해

## ■ 문제 분해의 예 : 타일 붙이기 문제

- 미장공이 3m, 2m의 벽에 가로와 세로가 각각 10cm인 타일을 붙이려고 함



- 타일 한 장을 붙이는 데 1g의 접착제 필요



- 필요한 타일의 수는  $30 \times 20 = 600$ 장



- 타일 한 장에 접착제 1g이 필요하므로 접착제를 총 600g 준비

## ■ 분해 정복(divide and conquer)

- 문제를 작은 문제로 분해하여 이를 해결하고, 해결된 작은 문제를 결합하여 큰 문제를 해결하는 방식
- 이진 탐색(binary search)
  - 분해 정복을 이해할 수 있는 가장 단순한 방법
  - 아래 그림은 7번 만에 탐색 종료

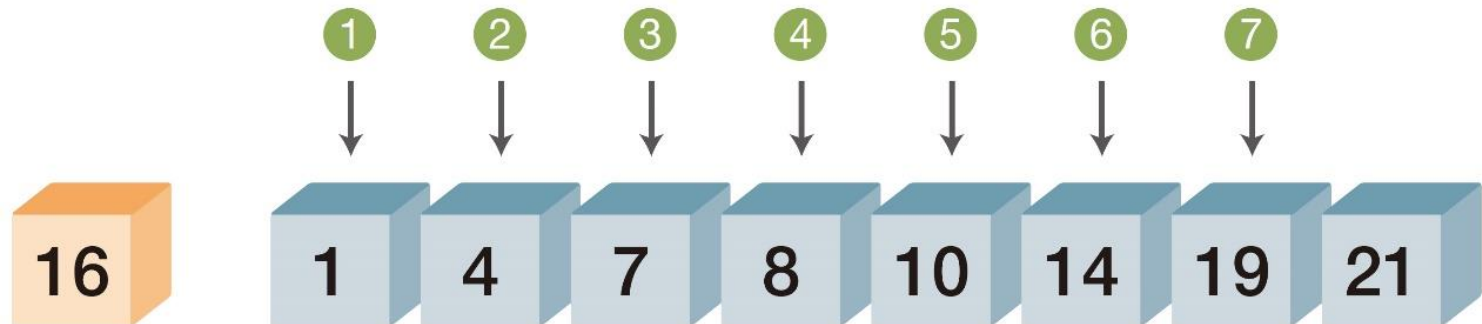


그림 7-16 일반적인 문제 해결 방식

- 이진 탐색(binary search)
  - 전체 숫자를 계속 반으로 나누어 비교, 3번 만에 탐색 종료

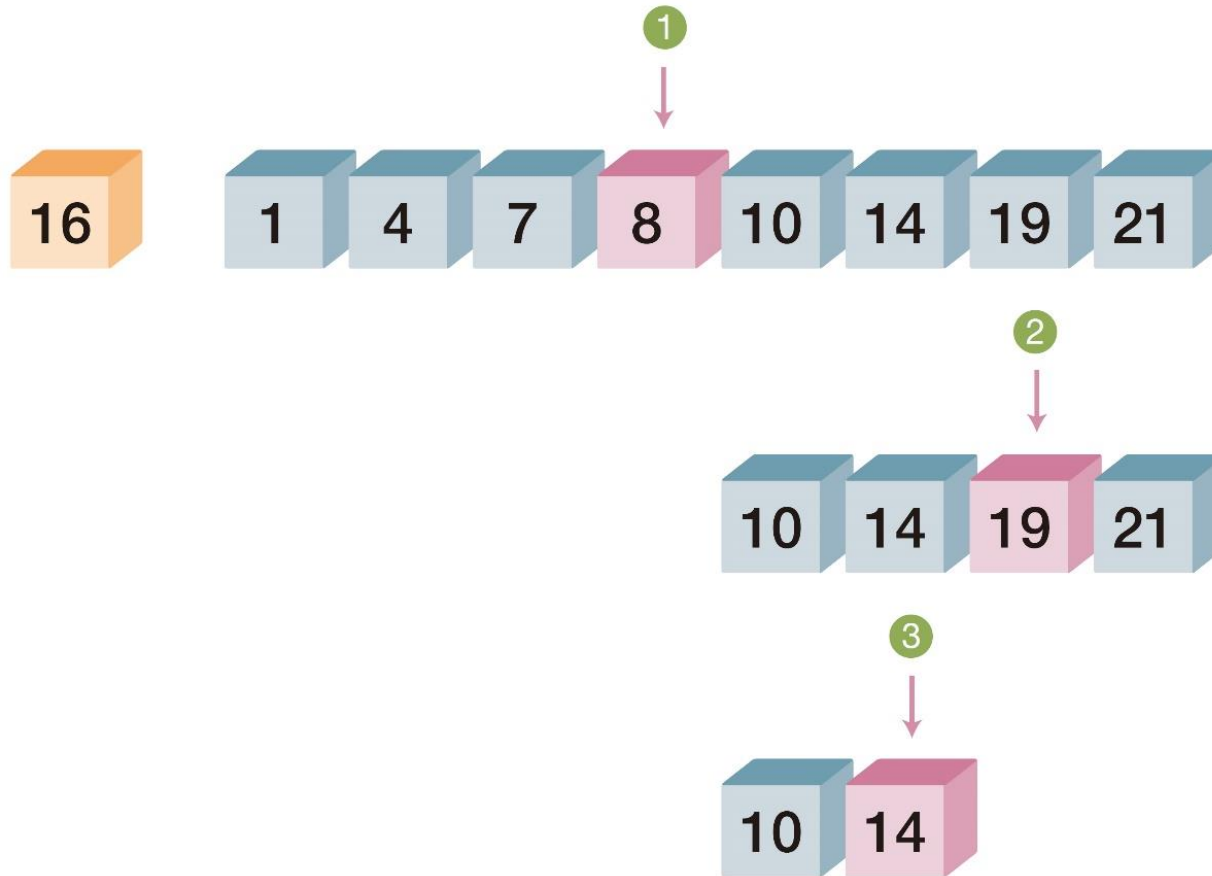


그림 7-17 분해 정복(이진 탐색) 문제 해결 방식

## ■ 패턴 인식의 정의

- 컴퓨팅 사고에서 패턴 인식은 문제에 적용 가능한 패턴을 찾아내는 과정
- 패턴을 찾을 수 있다면 문제를 해결하기 매우 쉬움
- 걷는 사람의 일정한 패턴
  - 오른발이 앞으로 나가면 왼팔은 뒤로, 왼발이 앞으로 나가면 오른팔은 뒤로 감



그림 7-18 서 있는 사람과 걷는 사람

## ■ RGB 패턴

- 빛을 이루는 3원색인 빨간색, 녹색, 파란색이 모든 사진이나 동영상의 패턴
- RGB 조합에 따라 16만 가지 이상의 색을 만들 수 있음
- 점들이 모여 사진이 되고, 사진을 빠르게 돌리면 동영상이 됨



그림 7-19 RGB 패턴

## ■ 수학의 패턴

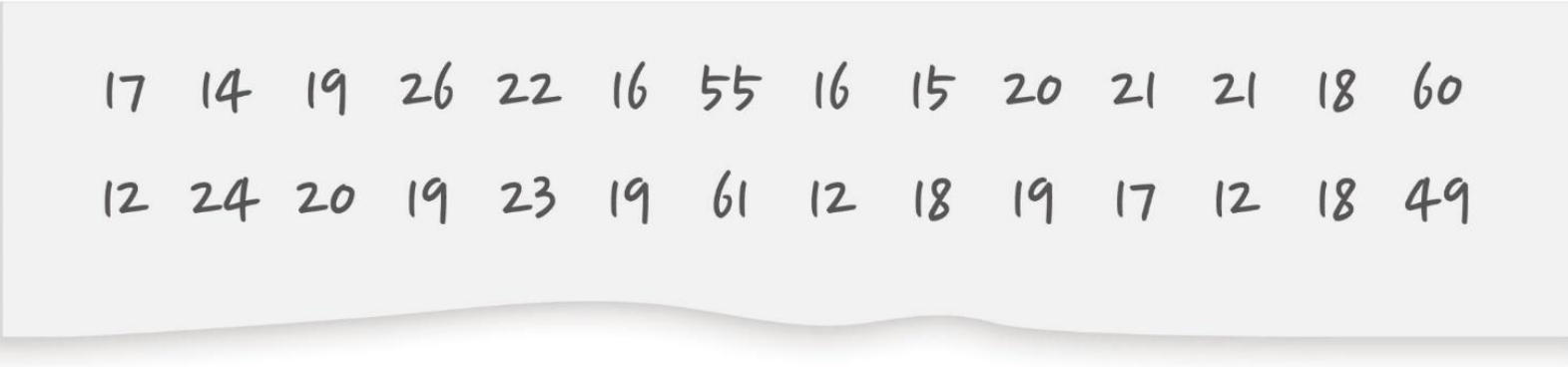
- 특정 수의 패턴을 찾는다면 문제를 해결하기 쉬워짐
- 맨 마지막 X에는 어떤 숫자가 와야 할까?

1, 2, 3, 5, 8, 13, 21, 34, X

그림 7-20 수열

## ■ 예) 어느 식당의 테이블 부족 문제

- 보통 날에는 테이블이 부족하지 않다가 특정 날에는 테이블이 부족하여 손님을 받지 못함
- 손님이 많이 몰릴 때를 대비하여 보조 테이블을 놓지만, 평상시 보조 테이블은 자리만 차지하고 쓸모가 없음



17 14 19 26 22 16 55 16 15 20 21 21 18 60  
12 24 20 19 23 19 61 12 18 19 17 12 18 49

**그림 7-21** 어느 식당의 방문자 수 데이터



## ■ 예) 어느 식당의 테이블 부족 문제

- 방문자 수를 기록한 데이터에 요일을 넣었더니
  - 일요일에 사람이 많이 몰린다는 것을 알 수 있었음
  - 일요일에만 보조 테이블을 놓는 방법으로 문제 해결

월	화	수	목	금	토	일	월	화	수	목	금	토	일
17	14	19	26	22	16	55	16	15	20	21	21	18	60
12	24	20	19	23	19	61	12	18	19	17	12	18	49

그림 7-22 요일을 넣어 패턴 찾기

## ■ 알고리즘의 개념

- 주어진 문제를 어떻게 해결할지 그 방법과 절차를 기술한 것
- 요리 절차를 적어놓은 레시피와 같음



그림 7-23 라면 끓이기 알고리즘

## ■ 알고리즘의 표현 방법

- 자연어, 순서도, 의사 코드, 프로그래밍 언어를 사용하여 표현



그림 7-24 알고리즘의 표현 방법

## ■ 자연어

- 사람들이 일상생활에서 사용하는 언어
- 컴퓨터가 사용하는 프로그래밍 언어와는 구별됨
- 자연어로 알고리즘을 표현하는 것은 매우 복잡

시작

X에 3, Y에 5를 대입한다.

X 값을 Z에 대입한다.

Y 값을 X에 대입한다.

Z 값을 Y에 대입한다.

X와 Y 값을 출력한다.

끝

그림 7-25 자연어 사용 예

## ■ 순서도(flow chart)

- 자연어로 파악하기 어려운 전체 구조 흐름을 파악하는 데 많이 사용
- 약속된 기호와 선을 사용하여 문제 해결 과정을 표현

표 7-1 순서도 기호

기호	명칭	의미
	단말	순서도의 시작과 끝을 의미한다.
	흐름선	각 기호를 연결하며, 순서도의 흐름을 나타낸다.
	처리	계산 등 자료의 연산 또는 처리를 나타낸다.
	준비	변수의 초기값, 기억 장소의 설정 등 작업의 준비 과정을 나타낸다.
	판단	조건을 판단하여 '예' 또는 '아니오'로 이동한다.
	입출력	자료의 입력과 출력을 나타낸다.
	출력	출력 장치를 통한 출력을 나타낸다.

## ■ 순서도의 장단점

- 장점 : 알고리즘 흐름을 빠르게 파악할 수 있음
- 단점 : 복잡한 프로그램을 순서도로 작성하기 까다로움

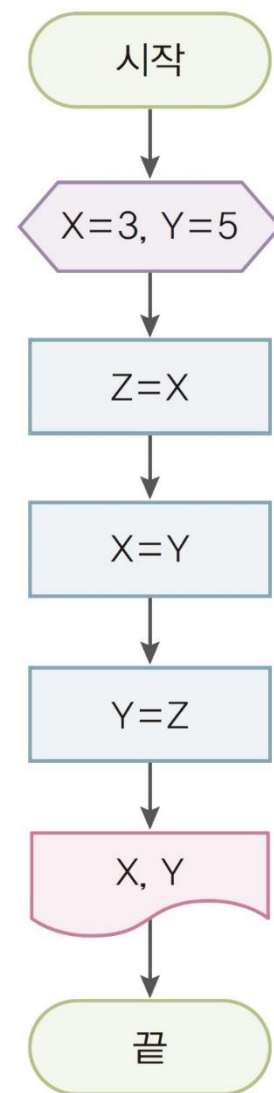


그림 7-26 순서도 사용 예

## ■ 의사코드(pseudo code)

- 특정 프로그래밍에 사용하는 언어와 유사한 서술로 알고리즘을 표현한 것
- 특정한 프로그래밍 언어의 문법을 따르지 않기 때문에 가짜 코드라는 의미로 의사코드라고 함
- 의사코드를 작성하면 특정한 프로그래밍 언어로 쉽게 변환 가능

```
START
  X=3, Y=5
  Z=X
  X=Y
  Y=Z
  PRINT X, Y
END
```

그림 7-27 의사코드의 사용 예

## ■ 알고리즘의 조건

- 입력 : 알고리즘에 입력되는 자료가 0개 이상 존재함
- 출력 : 알고리즘이 실행되면 결과 값이 1개 이상 나옴
- 유한성 : 알고리즘은 종료되어야 함
- 명확성 : 알고리즘의 명령이 모호하지 않고 명확해야 함
- 수행 가능성 : 알고리즘의 명령은 수행 가능해야 함



## ■ 알고리즘 설계란?

- 문제를 해결하기 위해 가장 효율적인 방법을 찾아내는 과정
- 알고리즘을 설계할 때에는 먼저 문제의 현재 상태와 목표 상태를 명확히 정의해야 함
- 현재 상태와 목표 상태를 정확히 인지해야 현재 상태에서 목표 상태로 도달하려면 어떤 작업을 수행해야 하는지 그 종류와 순서를 파악할 수 있음

## ■ 제어 구조

- 순차 구조 : 어떤 일을 처리하는 데 필요한 과정을 시간적인 순서에 따라 순차적으로 나타낸 구조로 보통 위에서 아래로 하나씩 실행
- 선택 구조 : 특정 조건을 만족하는지 아닌지에 따라 다음 명령을 선택적으로 실행하는 구조
- 반복 구조 : 어떤 문제를 해결하기 위해 동일한 동작을 반복적으로 실행하는 구조

## ■ 제어 구조

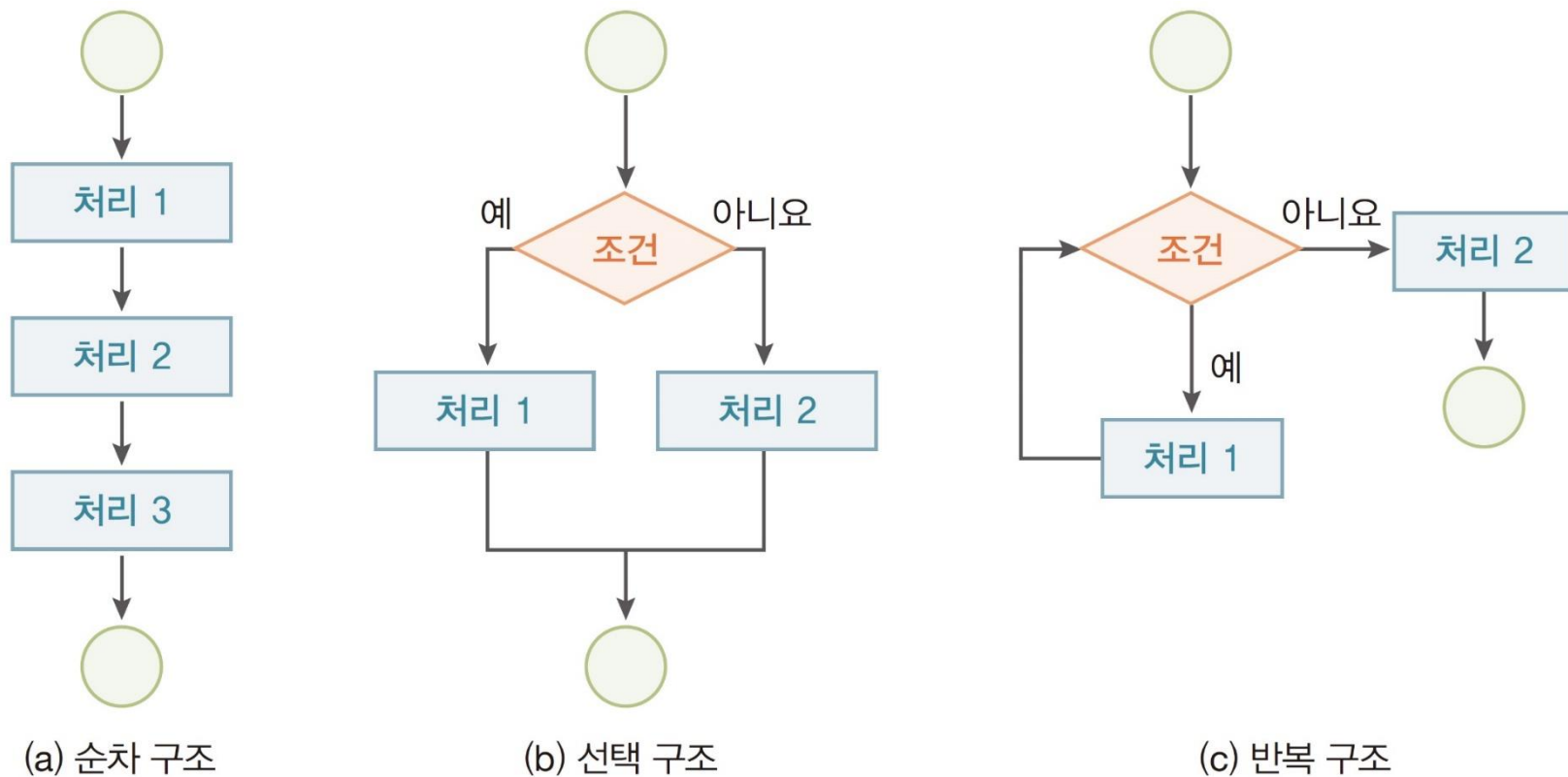
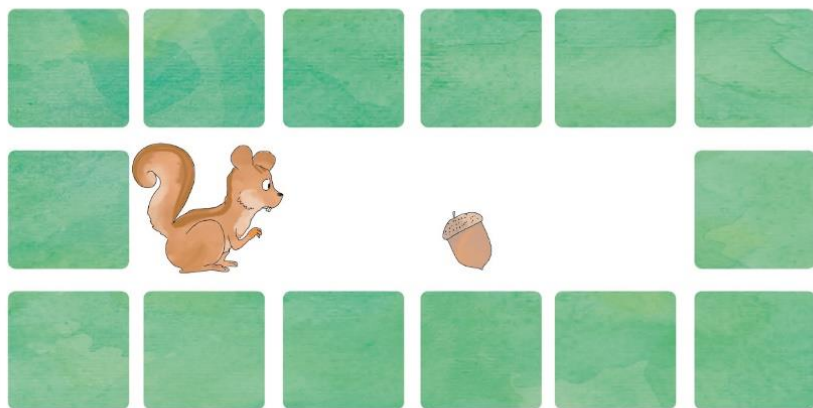
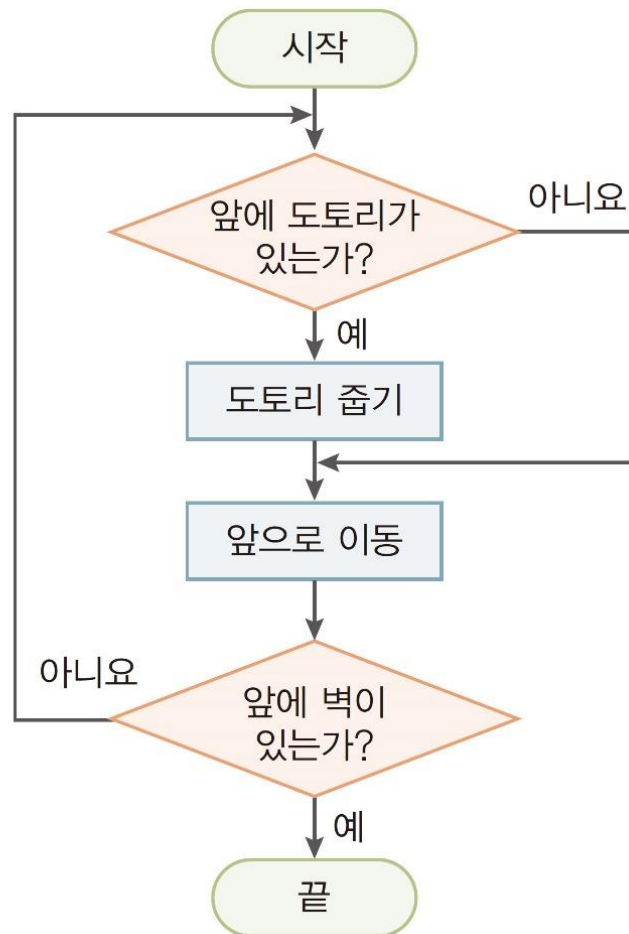


그림 7-28 제어 구조를 활용한 알고리즘

## ■ 도토리 줍기 알고리즘



(a) 밀폐된 공간의 다람쥐와 도토리



(b) 순서도 구현

그림 7-29 순서도로 도토리 줍기 알고리즘 설계

## ■ 알고리즘의 선택 기준

- 결과가 나올 때까지의 실행 시간이 짧은 것
- 컴퓨팅 기기의 기억 장소를 적게 사용하는 것

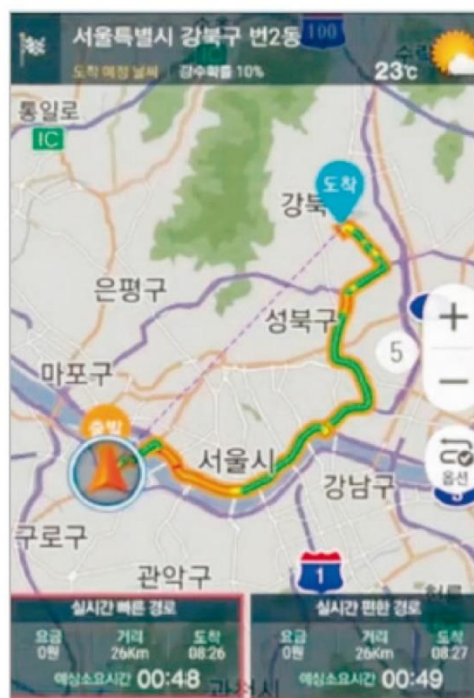


그림 7-30 내비게이션 비교

## ■ 알고리즘의 성능

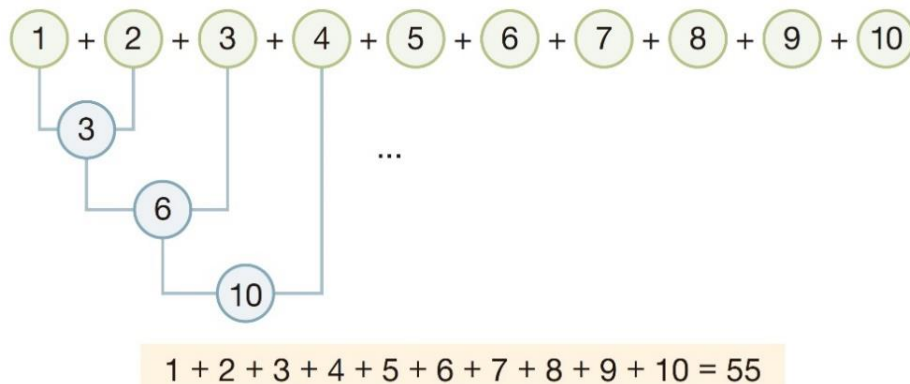
- 알고리즘의 실행 단계가 복잡하거나 처리해야 하는 자료가 많을 경우 알고리즘의 효율성은 프로그램의 성능에 중요한 역할을 함
- 알고리즘 성능 분석 방법
  - 알고리즘을 구현한 프로그램을 직접 실행하는 방법
  - 알고리즘의 실행 횟수 등 복잡도를 분석하는 방법
- 알고리즘의 복잡도
  - 알고리즘이 특정 기준에 따라 얼마나 빠르게 또는 느리게 실행되는지 나타내는 것
  - 시간 복잡도와 공간 복잡도를 분석하면 가장 효율적인 알고리즘을 선택할 수 있음

## ■ 시간 복잡도

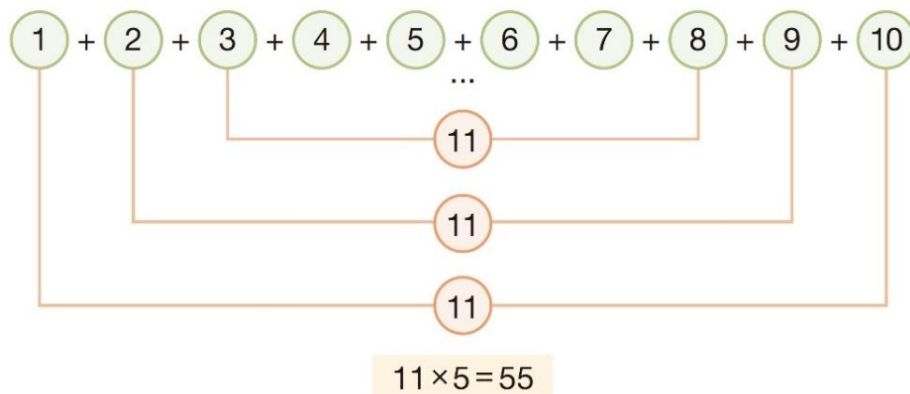
- 알고리즘이 실행되어 종료될 때까지 어느 정도의 시간이 필요한지 측정하는 방법
- 실제 컴퓨터의 실행 시간을 측정하기는 어렵기 때문에 알고리즘의 실행문이 몇 번 실행되는지 횟수를 표시하여 측정

## ■ 시간 복잡도

- ‘방법 2’가 ‘방법 1’보다 더 적은 연산으로 문제 해결하므로 시간 복잡도가 낮음



(a) 방법 1



(b) 방법 2

그림 7-31 1부터 10까지 수를 더하는 두 가지 알고리즘

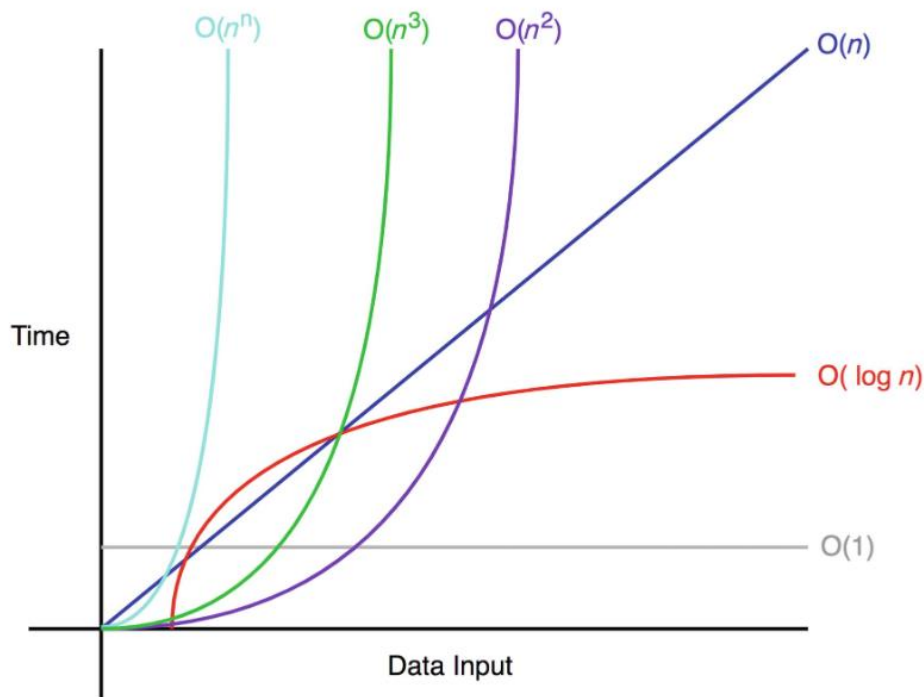


## 여기서 잠깐 빅오 표기법

빅오 표기법(Big O notation)은 알고리즘의 시간 복잡도를 표현하는 방법이다. 수학에서는 함수 성질을 나타낼 때 사용하며, 컴퓨터에서는 입력한 값의 크기에 따라 알고리즘 처리 횟수가 얼마나 증가하는지 나타낸다.

다음 두 경우를 비교하면  $O(n)$ 이  $O(n^2)$ 보다 시간 복잡도가 더 좋은 알고리즘이라는 것을 알 수 있다.

- $O(n)$ : 알고리즘의 수행 횟수도  $n$ 만큼 커진다.
- $O(n^2)$ : 알고리즘의 수행 횟수도  $n^2$ 만큼 커진다.



## ■ 공간 복잡도

- 알고리즘이 문제를 해결하는 데 어느 정도의 저장 공간을 필요로 하는지 측정하는 방법
- 기억 장치 내의 공간을 얼마나 적게 사용하는지가 중요
- 저장 공간은 알고리즘이 사용하는 공간과 알고리즘에 입력되어 처리되는 자료 공간을 모두 포함

## ■ 공간 복잡도

- ‘방법 2’의 알고리즘이 공간 복잡도가 더 낮아서 기억 장치의 공간을 효율적으로 활용

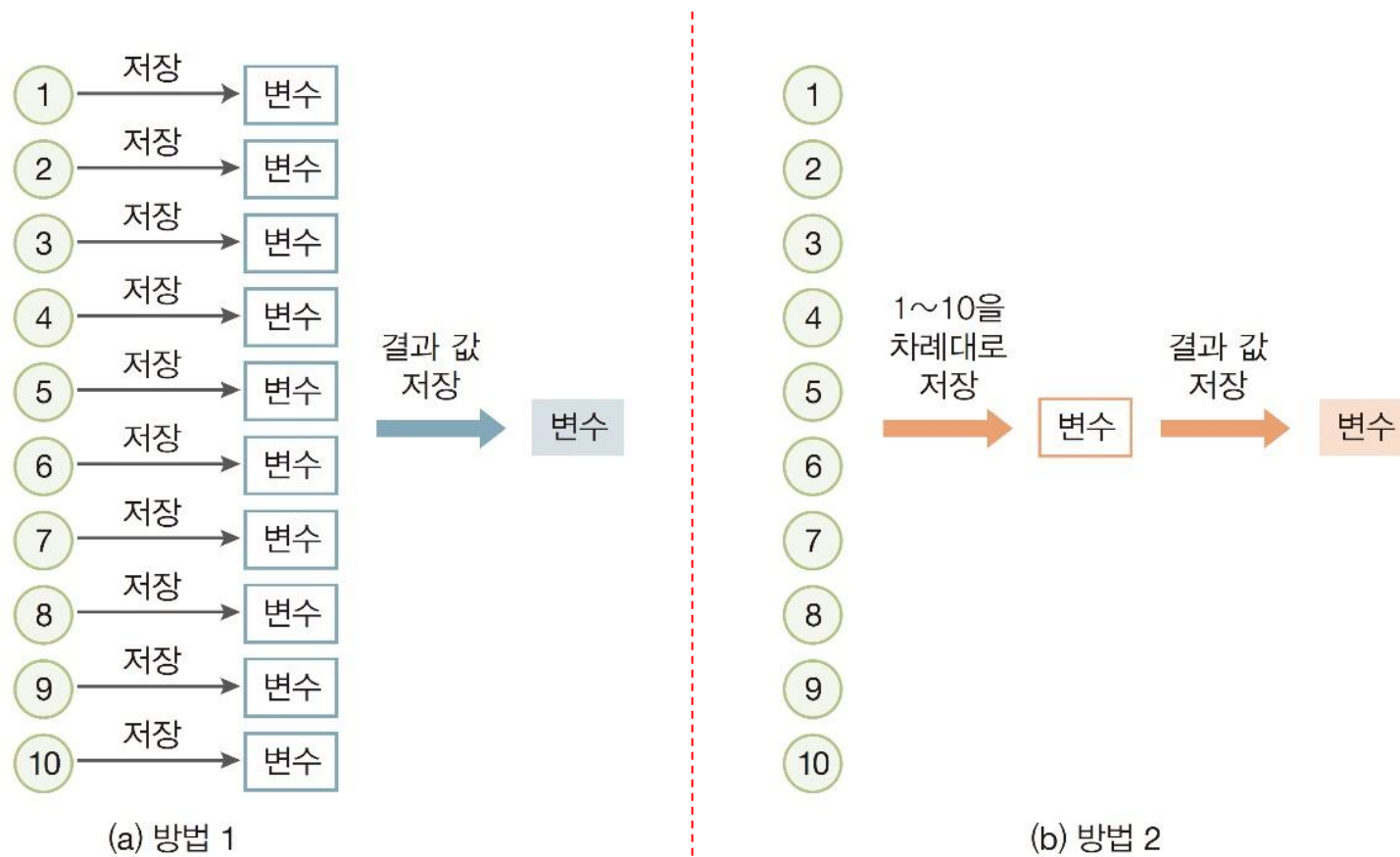


그림 7-33 1부터 10까지 수를 더하는 알고리즘에서 수를 저장하는 방법

## ■ 스무고개 문제 분해

- 출제자가 1에서 10까지 숫자 중 하나를 생각하면,  
참가자는 출제자가 생각한 숫자를 맞추는 게임



그림 7-34 스무고개 문제 분해

## ■ 스무고개 문제 분해

- 출제자와 참가자의 역할을 나누어 분해
- 참가자는 특정 숫자를 말하고 출제자는 자신이 생각하는 숫자가 맞을 때까지 '크다' 혹은 '작다'를 반복하는 패턴



그림 7-35 스무고개 게임 패턴

## ■ 스무고개 문제 분해

- 아래 작업을 계속 반복하면 의사코드가 만들어짐
  - 참가자 숫자가 출제자 숫자와 같으면 프로그램이 종료
  - 같지 않으면, 참가자 숫자가 출제자 숫자보다 큰 경우에는 ‘크다’고 출력
  - 같지도 크지도 않다면 참가자 숫자가 작은 경우는 딱 하나 ‘작다’고 출력
- 스무고개 게임에서 숫자를 맞추어 가는 과정은 이진탐색과 비슷함
  - 계속 중간 값을 불러 가능성이 없는 숫자를 없애는 것이 정답에 빠르게 접근할 수 있는 방법

## ■ 논리적인 접근 방법과 컴퓨팅 사고

- 알고리즘을 배우는 것은 문제를 푸는 논리적인 과정을 배우는 것

## ■ 알고리즘을 이용한 문제 해결 과정

- ❶ 코드 작성 : 의사코드 형태의 알고리즘 설계
- ❷ 코드 검토 : 의사코드를 머릿속에서 시뮬레이션
- ❸ 입력 및 실행 : 프로그래밍 언어로 프로그램 작성 후 실행,  
오류가 발생하면 ❶ 단계로 돌아가 코드 수정, ❶~❸ 단계 반복

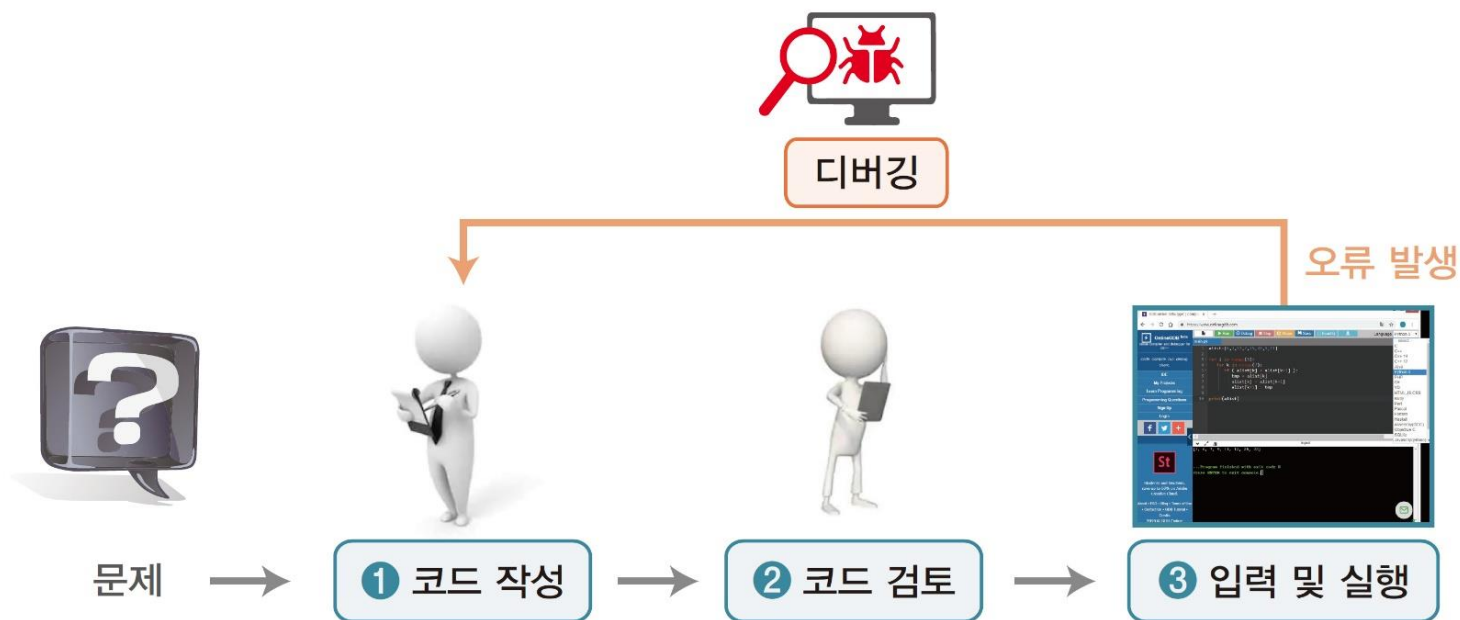


그림 7-36 알고리즘을 작성하여 문제를 해결하는 과정



## ■ 오류

- 문법 오류(syntax error)
  - 여러 가지 오류 중 가장 간단, 어떤 지점에서 오류가 발생했는지 알려 주기 때문
- 논리 오류
  - 문법적인 오류가 없는데도 원하는 답이 나오지 않는 오류가 가장 고치기 어려움
  - 알고리즘을 잘못 만들었기 때문에 발생

## ■ 프로그래밍 언어 (programming language)

- 컴퓨터에 작업을 지시하는 데 사용하는 언어
- 기계어, 어셈블리어, C, C++, 자바, 파이썬 등



그림 7-37 다양한 컴퓨터 언어들

## ■ 저급 언어(low level language)

### ■ 사람이 이해하기 힘든 언어

- 기계어(machine language) : 컴퓨터가 이해하는 언어로, 0과 1의 숫자로만 구성
- 어셈블리어(assembly language) : 기계어를 사람이 이해할 수 있는 문자 형태로 바꾸어 놓은 것



(a) 기계어



(b) 어셈블리어

그림 7-38 저급 언어

### ■ 고급 언어(high level language)

- 사람이 사용하는 단어로 이해하기 쉽게 만든 언어
- 기계어와 어셈블리어를 제외한 대부분의 언어가 고급 언어
- C 언어, java, 파이썬 등

## ■ 객체 지향 언어의 특징

- 객체 지향 언어는 데이터를 담는 통과 통에 담긴 데이터를 처리할 수 있는 함수를 하나로 묶어 놓은 것
- 변수와 함수를 하나로 묶어 객체로 처리하면 분리해서 사용할 때보다 편리하게 데이터를 처리할 수 있는 장점이 있음
- 다른 객체의 상속이나 데이터를 사용할 수 있는 사람과 데이터를 사용할 수 없는 사람을 구분하는 것 같은 추가 기능도 있음



그림 7-39 일반 언어와 객체 지향 언어의 차이

## ■ 객체 지향 언어 java

- 객체 지향 언어 중 현재 가장 많이 사용하는 언어
- 호환성 문제를 해결한 언어
- 자바로 프로그래밍하면 대부분의 운영체제에서 작동하기 때문에 코드를 수정할 필요가 없음

## ■ 가상머신

- 운영체제와 응용 프로그램 사이에서 작동하는 프로그램
- 가상머신을 설치하면 응용 프로그램이 모두 동일한 환경에서 작동하는 것처럼 보임
- 자바로 만들어진 프로그램을 사용하기 위해서는 자바 가상머신(Java Virtual Machine, JVM)을 설치해야 함

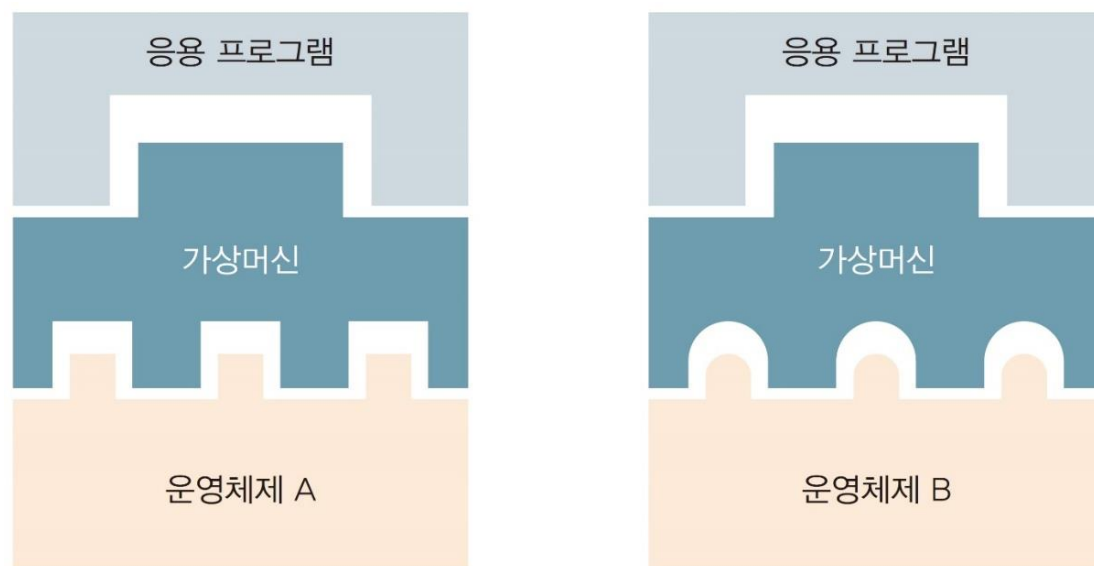


그림 7-40 가상머신의 개념

## ■ 가상머신의 활용

- 윈도우 운영체제 환경에서 유닉스를 사용하고 싶을 때는 윈도우 운영체제에 유닉스 가상머신을 설치하여 사용(맥 OS와 윈도우 동시 사용 가능)
- 가상머신을 사용하면 호환성이 높지만, 응용 프로그램이 가상머신에서만 작동하기 때문에 느림



## ■ 언어 번역 방식

- 알기 쉽게 만든 언어를 고급 언어라고 하며, 고급 언어로 짠 코드를 소스코드라고 함
- 컴퓨터는 기계어만 인식하기 때문에 소스코드를 기계어로 번역해야 함
- 대표적인 언어 번역 방식
  - 컴파일러(compiler)
  - 인터프리터(interpretor)

## ■ 컴파일러

- 소스 코드를 컴퓨터가 실행할 수 있는 기계어로 번역하여 실행 파일을 만든 후 한꺼번에 실행(C 언어, 자바 등)

## ■ 인터프리터

- 소스 코드를 한 번에 한 행씩 번역하여 실행(자바스크립트, 파이썬 등)



그림 7-41 컴파일러와 인터프리터

## ■ 인터프리터와 컴파일러의 레시피 비유

- 인터프리터 : 레시피에 적힌 대로 위에서 아래로 한 줄씩 읽어 가며 요리를 완성
- 컴파일러 : 요리를 시작하기 전에 레시피를 검토한 후 오류나 불필요한 내용 확인

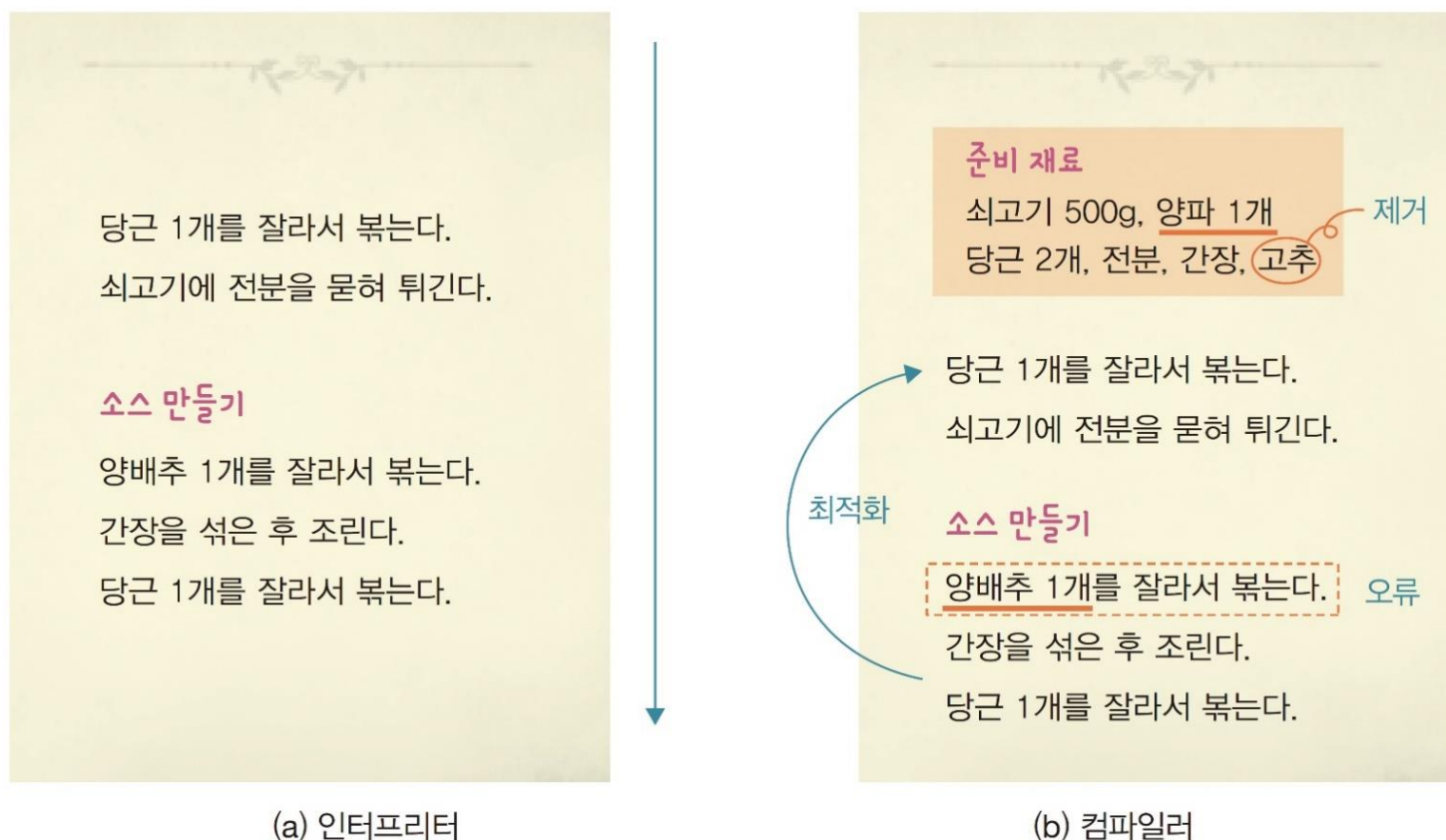


그림 7-42 인터프리터와 컴파일러의 차이

## ■ 컴파일러를 사용하는 목적

- 소스 코드에서 오류를 발견하여 실행할 때 문제가 없도록 하는 것
- 최적화

표 7-2 자바(컴파일러)와 자바스크립트(인터프리터) 차이

구분	자바	자바스크립트
변수	변수를 선언해야 한다.	변수를 선언할 필요가 없다.
실행	컴파일 후 실행된다.	한 줄씩 실행된다.
장점	오류 찾기와 코드 최적화, 분할 컴파일을 사용하여 공동 작업이 가능하다.	실행이 편리하다.
사용 프로그램	대형 프로그램	간단한 프로그램

**Thank you!**