05 강

컴퓨터과학 개론

알고리즘 (1)

컴퓨터과학과 이관용 교수

학습목차

1 알고리즘의 개념

2 알고리즘의 설계

3 알고리즘의 분석

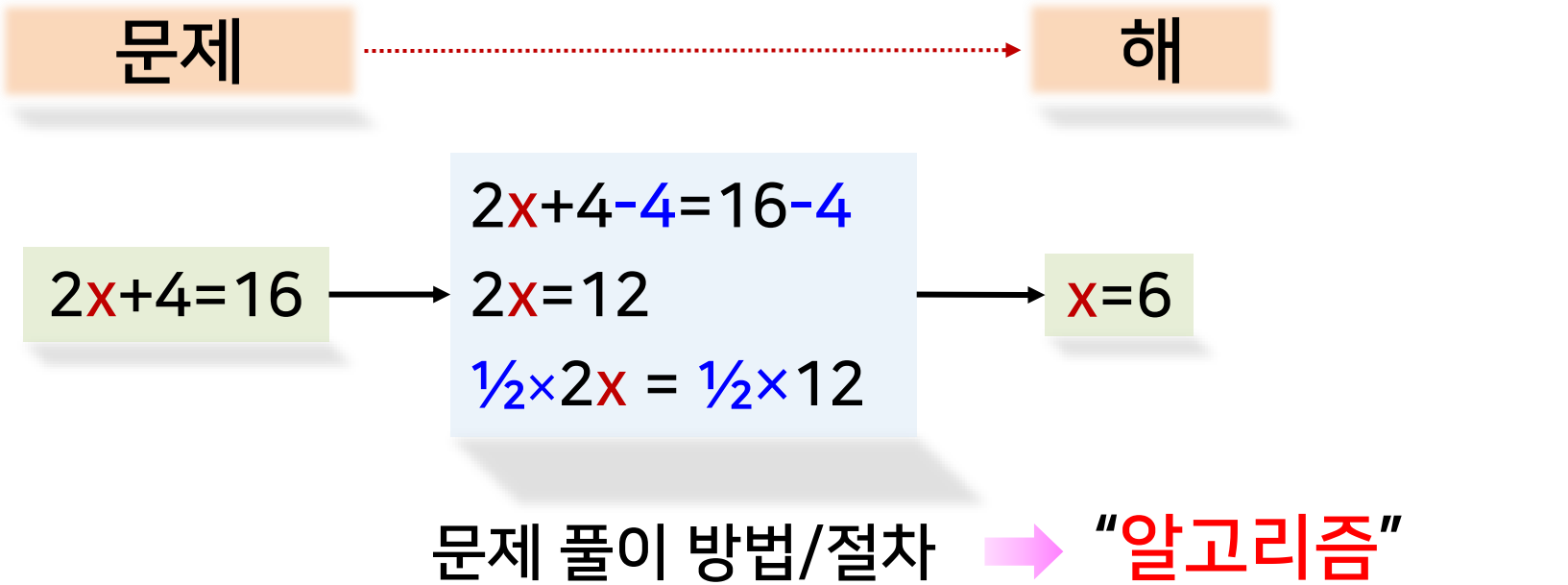
4 정렬 알고리즘: 선택정렬,버블정렬,삽입정렬

01

알고리즘의 개념

문제를 해결하려면

- **컴퓨터과학 → "컴퓨터를 활용해서 주어진 문제를 해결하는 학문"**



알고리즘?

문제 해결을 위한 "레시피"

- 단계적인 조리 절차("레시피")를 따르면 음식을 만들 수 있듯이,

  주어진 문제도 단계적인 풀이 절차("알고리즘")를 따르면
  문제의 해를 구할 수 있음

- "맛있고 좋은 음식" ↔ "효율적인 알고리즘"

알고리즘의 정의

주어진 문제를 풀기 위한 명령어들을 단계적으로 나열한 것

- 입출력 → 0개 이상의 외부 입력, 1개 이상의 출력 생성
- 명확성 → 각 명령은 모호하지 않고 단순 명확해야 함
- 유한성 → 한정된 수의 단계를 거친 후에는 반드시 종료해야 함
- 유효성 → 모든 명령은 컴퓨터에서 실행할 수 있어야 함

주어진 문제에 대한 결과를 생성하기 위해
모호하지 않는 간단하고 컴퓨터가 수행 가능한
일련의 유한개의 명령을 순서에 따라 구성한 것

- 실용적 측면 → 효율적이어야 함

알고리즘 생성 단계

설 계 → 표현/기술 → 정확성 검증 → 효율성 분석

방법

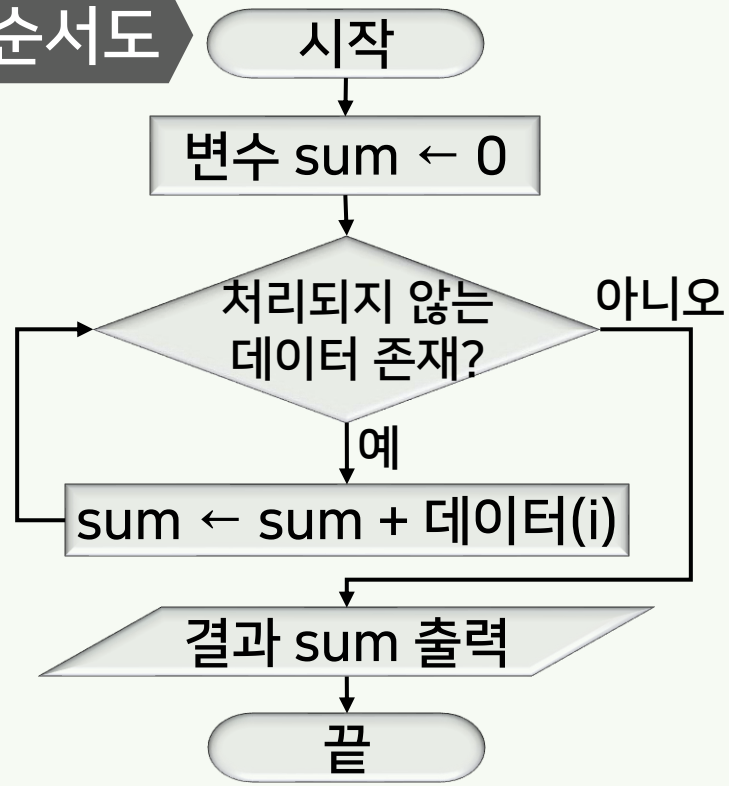
일상 언어

단계1. 계산할 n개의 숫자를 입력받음
단계2. 입력 데이터를 모두 더해 합을 계산
단계3. 합을 출력

의사코드

```
sum = 0;
while ( 데이터가 더 존재 )
   sum = sum + 데이터(i);
end
print(sum);
```

순서도



자료구조와 알고리즘의 관계

자료구조

- 데이터 사이의 논리적 관계를 표현하고 조직화하는 방법
- 기본적/대표적 종류 → 배열, 연결 리스트, 스택, 큐, 트리, 그래프

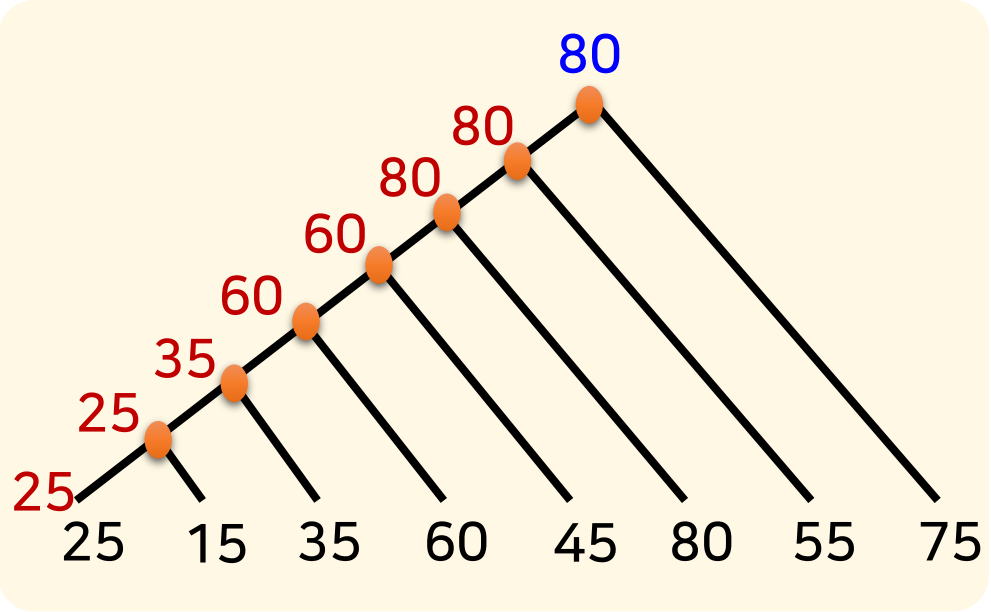
"효율적" 프로그램 ← 자료구조 + 알고리즘

- 자료구조에 대한 고려 없는 효율적인 알고리즘의 선택,
  알고리즘에 대한 고려 없는 효율적인 자료구조의 선택은 무의미

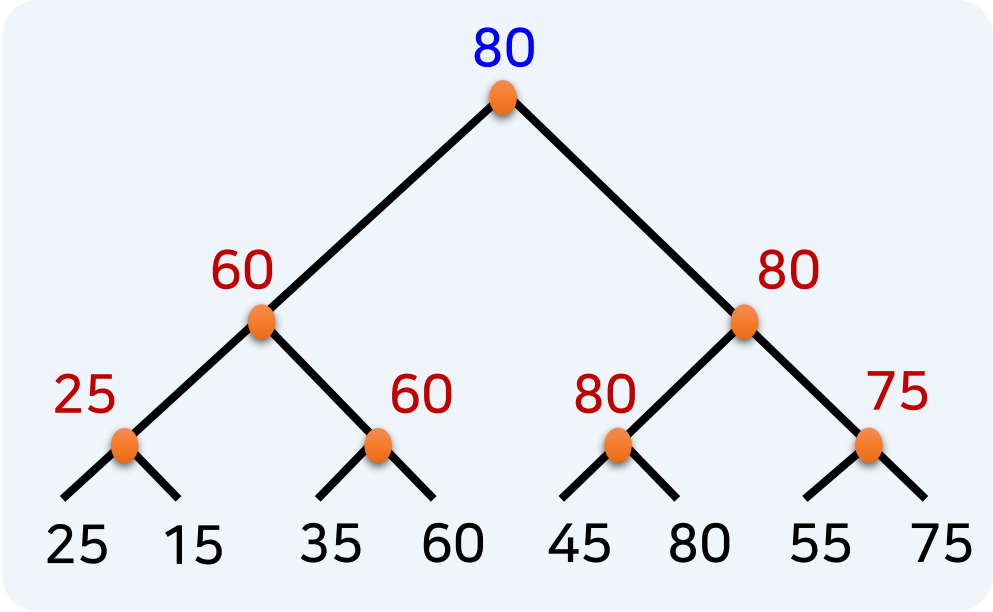
02

알고리즘의 설계

최댓값을 찾는 알고리즘



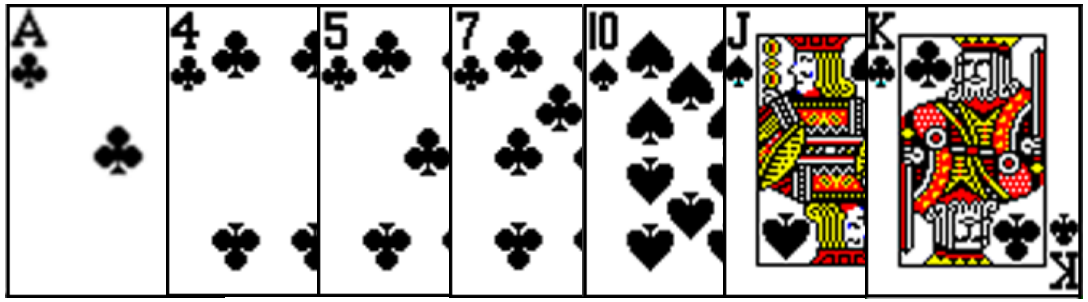
알고리즘1



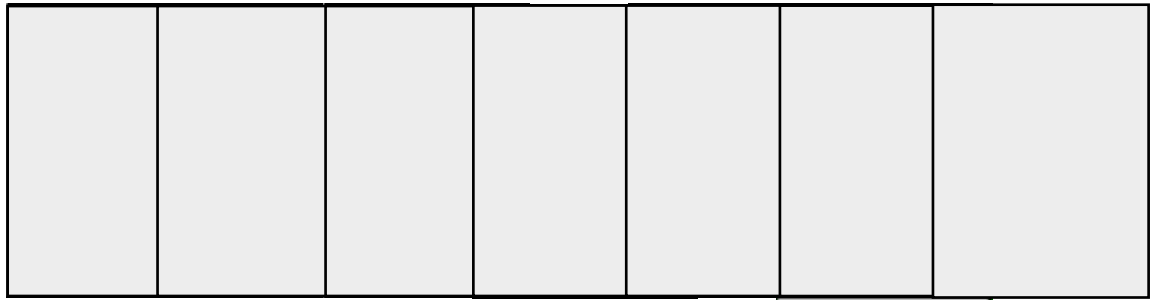
알고리즘2

최댓값 찾기에서 알고리즘1과 알고리즘2 중에서
어떤 것이 더 효율적인가?

뒤섞인 카드 중에서 원하는 카드 찾기

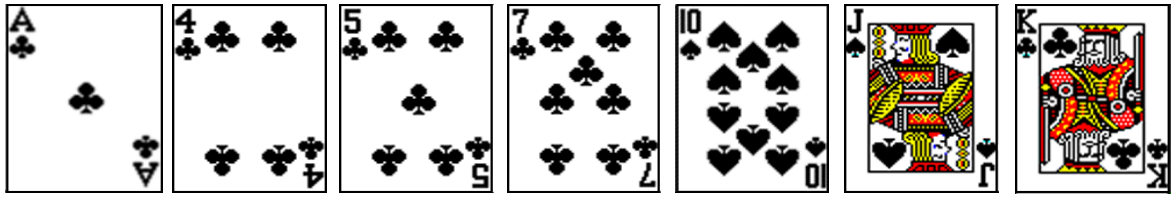


주어진 카드를 섞어서 뒤집어 놓았다. 이 중에서 K를 찾아라!

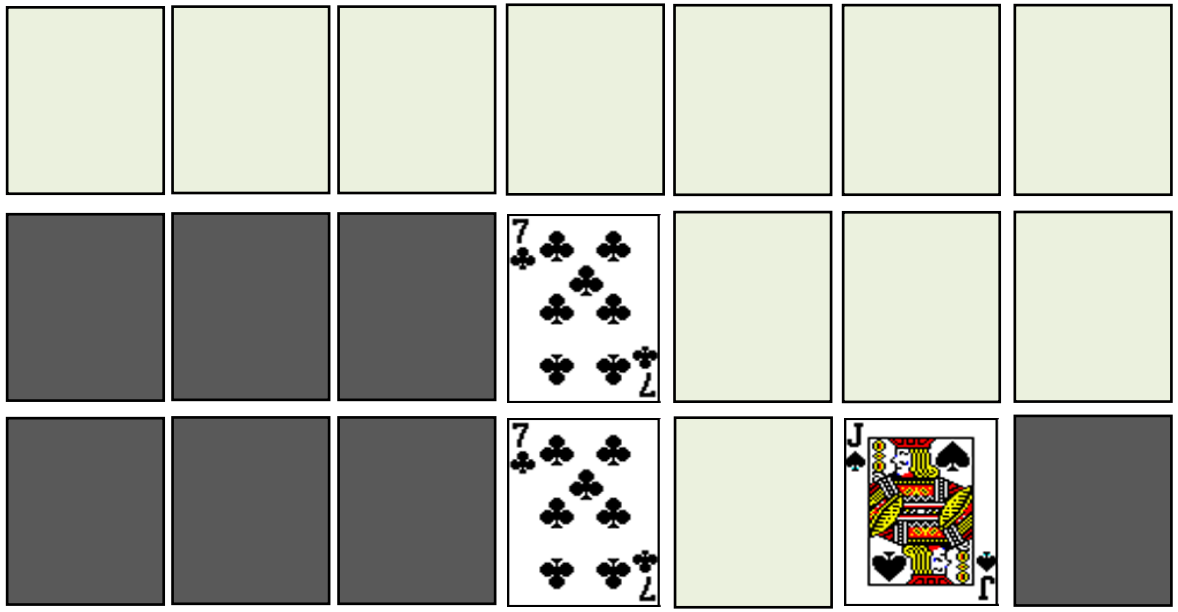


순차 탐색
sequential search

순서대로 나열된 카드 중에서 찾기



주어진 카드 중에서 10을 찾아라!



이진 탐색
binary search

알고리즘 설계에 적용할 방법?

주어진 문제와 그에 따른 조건 등이 매우 다양

→ 일반적이고 범용적인 설계 기법은 없음

대표적인 설계 기법

- 분할정복 divide-and-conquer 방법
- 동적 프로그래밍 dynamic programming 방법 ("동적 계획법")
- 욕심쟁이 greedy 방법 ("탐욕적 방법")

분할정복 방법

순환적으로 recursively 문제를 푸는 방법

- 문제의 입력을 더 이상 나눌 수 없을 때까지
  2개 이상의 작은 문제로 순환적으로 분할하고,
  분할된 문제들을 각각 해결한 후
  이들의 해를 결합하여 원래 문제의 해를 구하는 하향식 접근 방법

특징

- 분할된 작은 문제는 원래 문제와 동일, 단 입력 크기만 작아짐
- 분할된 작은 문제는 서로 독립적

분할정복 방법

각 순환 호출 시 다음 세 단계의 작업을 거침

분할 주어진 문제를 여러 개의 작은 문제로 분할

정복 작은 문제를 순환적으로 분할.
만약 작은 문제가 더 이상 분할되지 않을 정도로
크기가 충분히 작다면 순환호출 없이 작은 문제의 해를 구함

결합 작은 문제에 대해 정복된 해를 결합하여
원래 문제의 해를 구함 ※ 결합 단계가 없는 문제도 존재

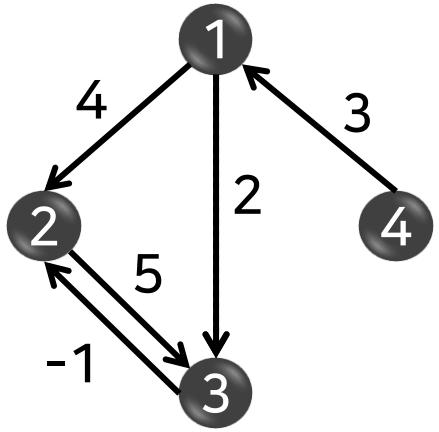
- 퀵 정렬, 합병 정렬, 이진 탐색

동적 프로그래밍 방법

최적화 문제의 해(최댓값, 최솟값)를 구하기 위한 상향식 접근 방법

- 문제의 크기가 작은 소문제에 대한 해를 구해서 테이블에 저장해 놓고, 이를 이용하여 크기가 보다 큰 문제의 해를 점진적으로 만들어 가는 방법
  - 소문제들이 서로 독립일 필요 없음

- 플로이드 알고리즘 (모든 정점 간의 최단 경로를 구하는 알고리즘)



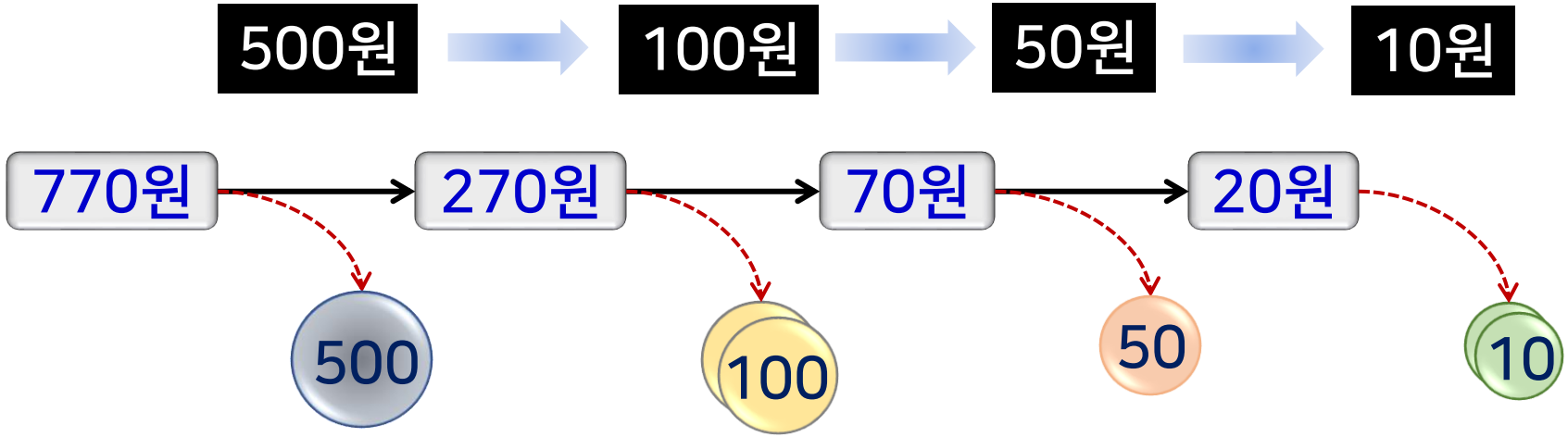
$$D = \begin{bmatrix} 0 & 1 & 2 & \infty \\ \infty & 0 & 5 & \infty \\ \infty & -1 & 0 & \infty \\ 3 & 4 & 5 & 0 \end{bmatrix}$$

욕심쟁이 방법

- **해를 구하는 일련의 선택 과정마다 전후 단계의 선택과는 상관없이 각 단계에서 '가장 최선'이라고 여겨지는 국부적인 최적해를 선택해 나가면 결과적으로 전체적인 최적해를 얻을 수 있을 것이라고 희망하는 방법**
  - 희망 → "각 단계의 최적해를 통해 전체적인 최적해를 만들어 내지 못할 수 있음"
    - 적용 범위가 제한적, 간단하면서 강력한 설계 기법
  - 거스름돈 문제, 배낭 문제

거스름돈 문제

- **고객에게 돌려줄 거스름돈이 T만큼 있을 때 고객이 받을 동전의 개수를 최소로 하면서 거스름돈을 돌려주는 방법을 찾는 문제**
  - 동전의 종류 → 500원, 100원, 50원, 10원



배낭 문제

- **최대 용량 M인 하나의 배낭, n개의 물체**
  - 각 물체 i에는 물체의 무게 w_i와
    해당 물체를 배낭에 넣었을 때 얻을 수 있는 이익 p_i

- **배낭의 용량을 초과하지 않는 범위에서**
  배낭에 들어 있는 물체의 이익의 합이 최대가 되도록
  배낭에 물체를 넣는 방법을 찾는 문제
  - 가정 → 물체를 쪼개서 넣을 수 있음

배낭 문제

용량 10kg

4kg, 이익 14

3kg, 이익 15

5kg, 이익 20

3kg, 이익 9

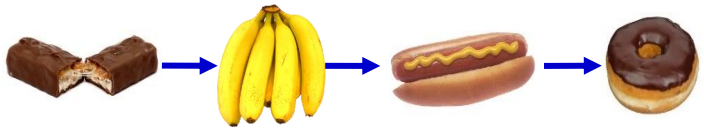
M = 10, n = 4

$(p_1, p_2, p_3, p_4) = (14, 15, 20, 9)$

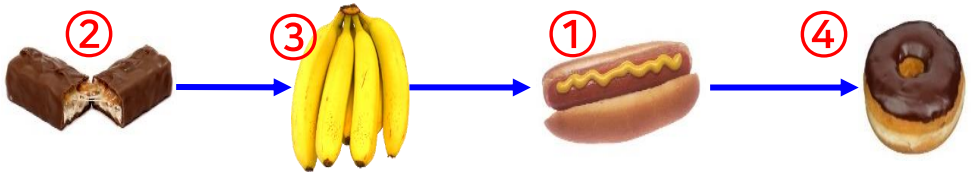
$(w_1, w_2, w_3, w_4) = (4, 3, 5, 3)$

단위 무게당 이익

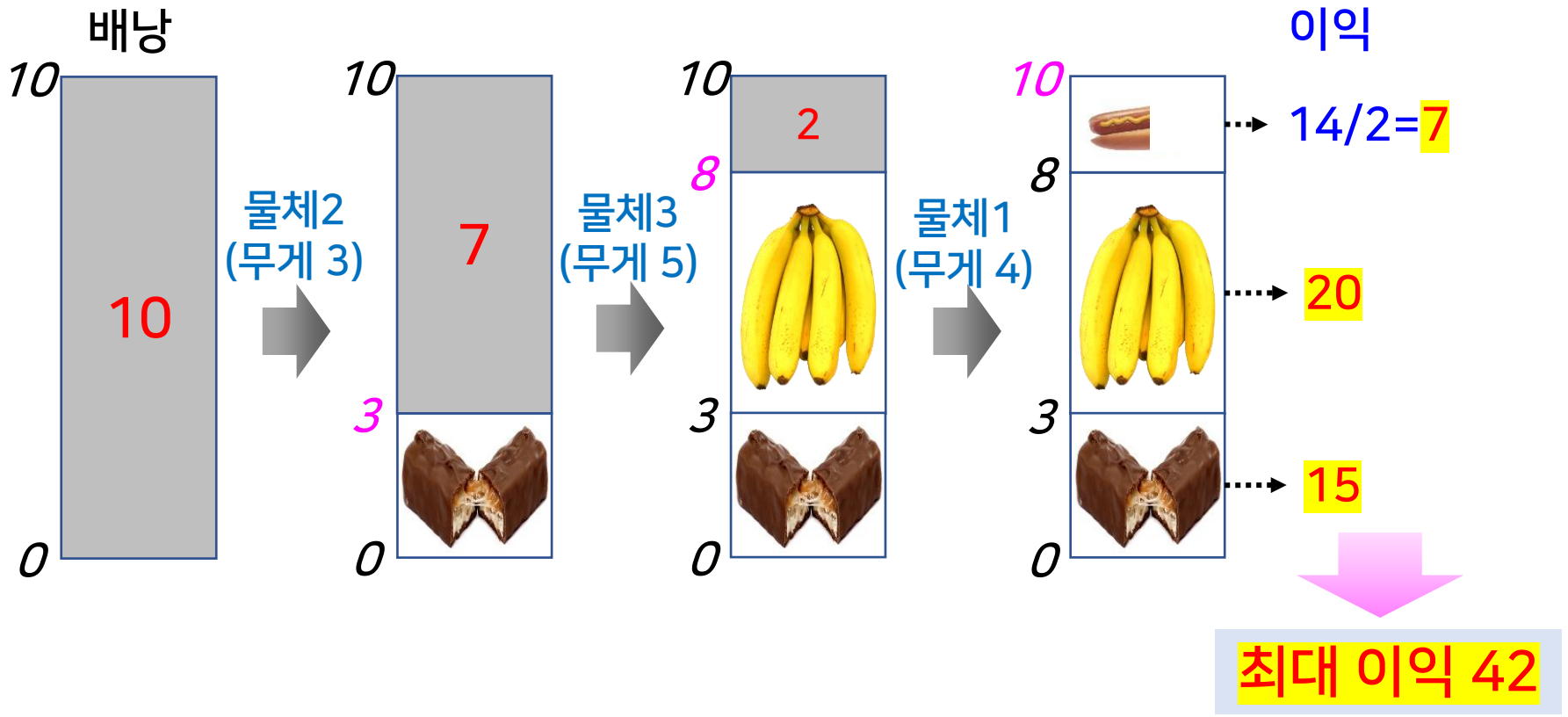
$$\left(\frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \frac{p_4}{w_4}\right) = (3.5, 5, 4, 3)$$



배낭 문제



3kg, 이익 15 5kg, 이익 20 4kg, 이익 14 3kg, 이익 9



03

알고리즘의 분석

알고리즘 분석

정확성 분석

- 유효한 입력과 유한 시간 → 정확한 결과 생성?
  - 다양한 수학적 기법을 사용한 이론적 증명이 필요

효율성 분석

- 알고리즘 수행에 필요한 컴퓨터 자원의 양을 측정
- 공간 복잡도 space complexity
  - 메모리의 양 = 정적 공간 + 동적 공간
- 시간 복잡도 time complexity
  - 수행 시간

시간 복잡도

알고리즘의 수행 시간

- 컴퓨터에서 실행시켜 완료될 때까지 걸리는 실제 시간을 측정하는 방법
  - 컴퓨터의 종류와 속도, 프로그래밍 언어, 프로그램 작성 방법, 컴파일러의 효율성 등에 종속적 → 일반성 결여

- "알고리즘에서 수행되는 단위 연산의 수행 횟수를 모두 더한 값"
  - 입력으로 제공되는 데이터의 크기("입력 크기")가 증가하면 수행 시간도 증가

    → 단순히 수행되는 단위 연산의 개수의 합이 아닌 입력 크기의 함수로 표현

  - 입력 데이터의 상태에 따라 달라짐

    → 평균 수행 시간, 최선 수행 시간, 최악 수행 시간

시간 복잡도

```
SumAverage( a[], n )
{ //a[0..n-1], n : 입력 배열과 데이터 개수
1   i = 0;
2   sum = 0;
3   while ( i < n ) {
4     sum = sum + a[i];
5     i = i + 1;
    }
6   average = sum / n;
7   print sum, average;
}
```

입력 크기

1
1
n+1
n
n
1
1

$f(n) = 3n+5$ → 점근성능 → $O(n)$

점근성능

입력 크기 n이 충분히 커짐에 따라 결정되는 성능

	$f_1(n)=10n+9$	$f_2(n)=n^2/2+3n$
n=5	59	27.5
n=10	109	80
n=15	159	157.5
n=16	169	176
n=20	209	260
⋮		

다항식의 수행 시간에서 가장 큰 영향을 미치는 것은?

- 계수 없이 최고차항만을 이용해서 간략히 표현하는 성능
  - 수행 시간의 어림값, 수행 시간의 증가 추세 → 알고리즘의 우열 비교가 용이

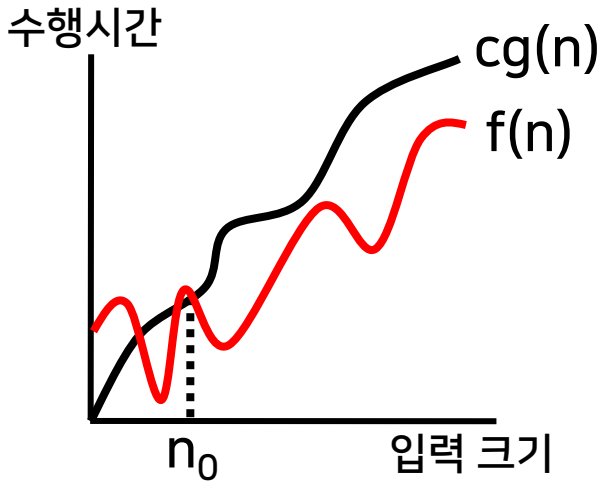
점근성능의 표기법

정의 1 'Big-oh' 점근적 상한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

$n \geq n_0$인 모든 n에 대하여 $f(n) \leq c \cdot g(n)$을 만족하는

양의 상수 c와 n_0이 존재하면 $f(n)=O(g(n))$이다.



$$f(n) = O(g(n))$$

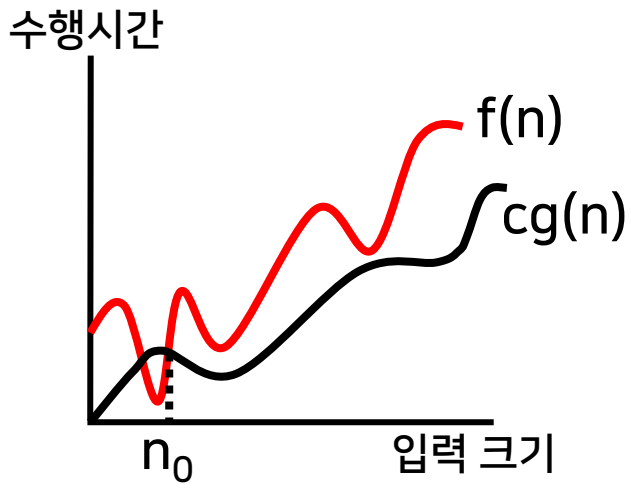
점근성능의 표기법

정의 2 'Big-omega' 점근적 하한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

$n \geq n_0$인 모든 n에 대하여 $f(n) \geq c \cdot g(n)$을 만족하는

양의 상수 c와 n_0이 존재하면 $f(n)=\Omega(g(n))$이다.



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

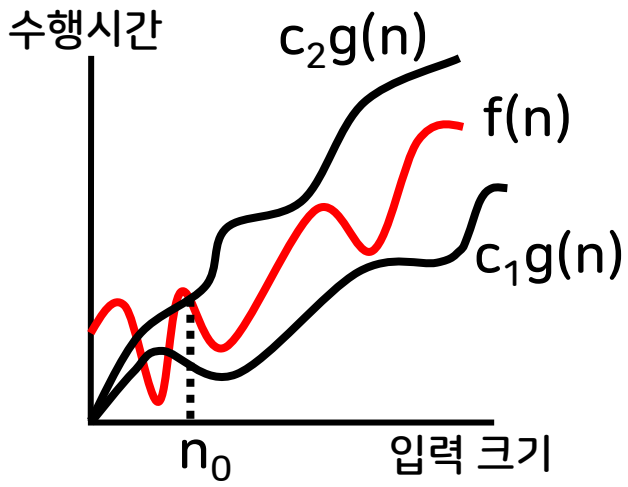
점근성능의 표기법

정의 3 'Big-theta' 점근적 상하한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

$n \geq n_0$인 모든 n에 대하여 $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$을 만족하는

양의 상수 c_1, c_2와 n_0이 존재하면 $f(n)=\Theta(g(n))$이다.



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

$$f(n) = O(g(n)) = \Omega(g(n))$$

점근성능의 표기법

- **$f(n)=3n+3,\ g(n)=n$**
  - $n \geq n_0$에 대해서 $f(n) \leq c \cdot g(n)$을 만족 → $n_0=2$, $c=5$ → $f(n)=O(g(n))=O(n)$
  - $n \geq n_0$에 대해서 $f(n) \geq c \cdot g(n)$을 만족 → $n_0=1$, $c=3$ → $f(n)=O(g(n))=\Omega(n)$
  - $f(n)=O(n)$ 이면서 $f(n)=\Omega(n)$ → $f(n)=\Theta(n)$

- **$f(n)=2n^3+3n^2+n+10$** → $O(n^3)$

O-표기 간의 연산 시간의 크기 관계

상수 시간 로그 시간 선형 시간 로그 선형 시간 제곱 시간 세제곱 시간 지수 시간

$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

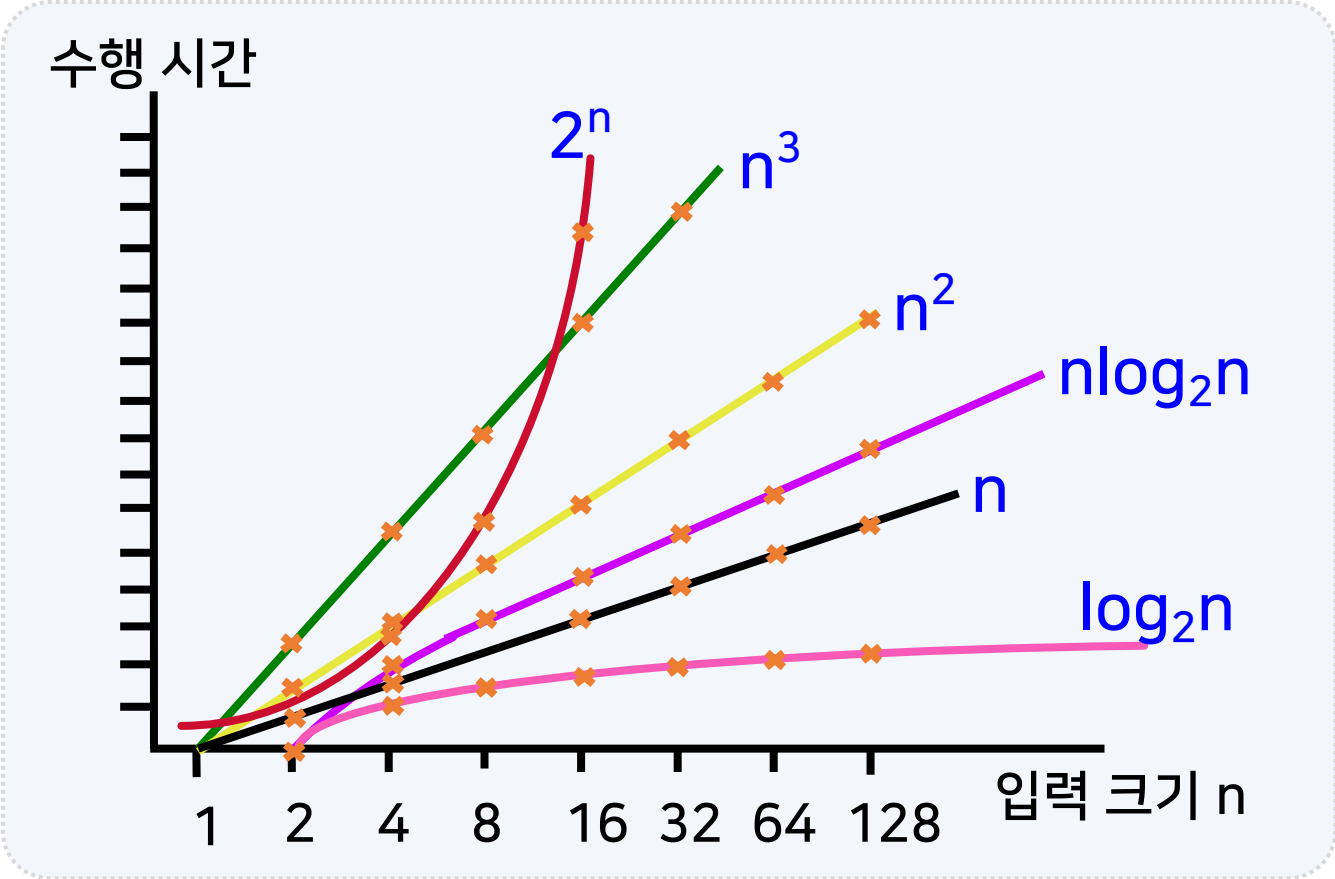
효율적 ⟵ ⟶ 비효율적

$f_1(n)=10n+9$ → $O(n)$

$f_2(n)= n^2/2+3n$ → $O(n^2)$

$f_3(n)= 3n^3+3n+2$ → $O(n^3)$

O-표기 간의 연산 시간의 크기 관계

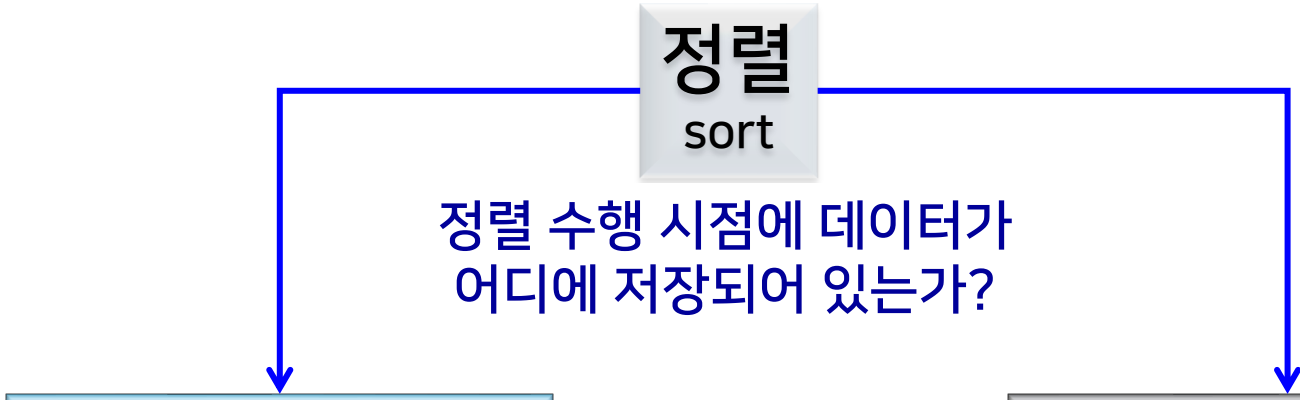


04

정렬 알고리즘:

선택 정렬, 버블 정렬, 삽입 정렬

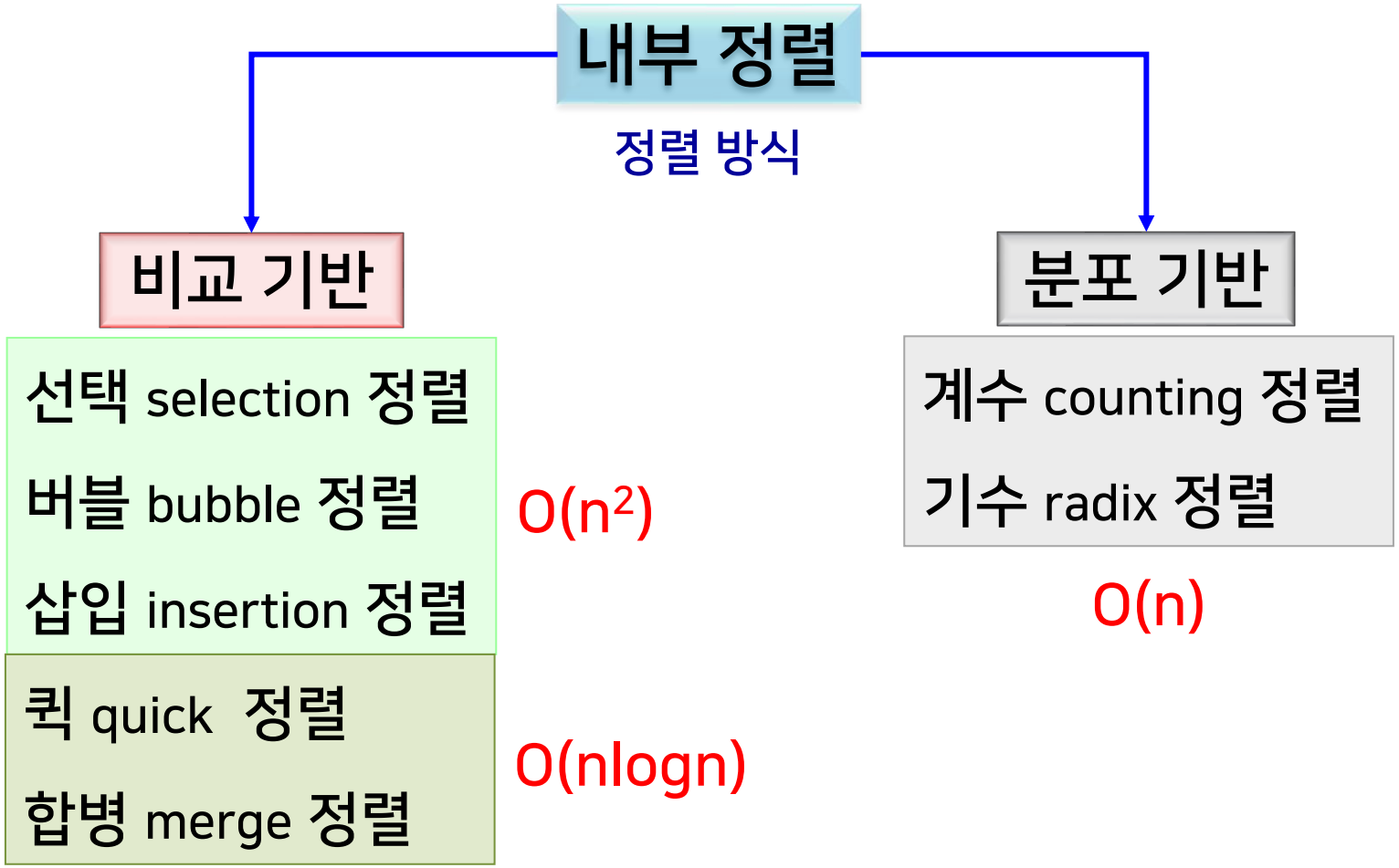
기본 개념



모든 데이터를
주기억장치에 적재한 후
정렬하는 방식

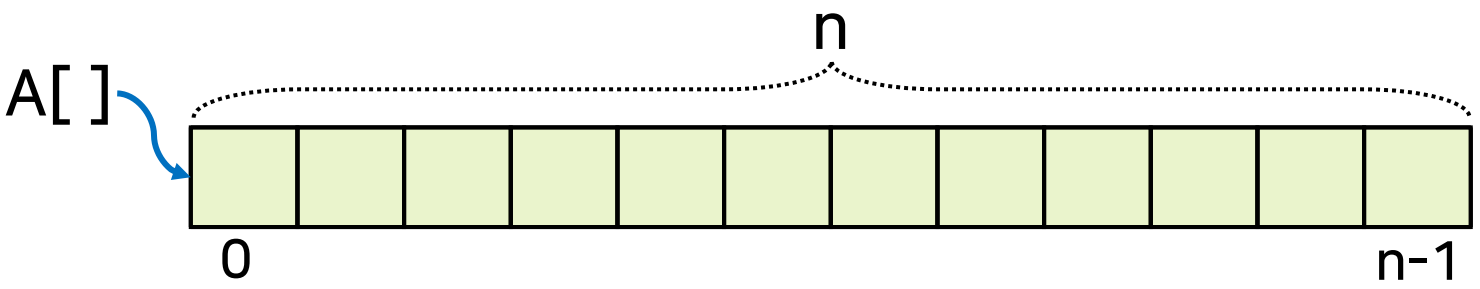
모든 데이터를 주기억장치에 저장할 수 없는 경우,
전체 데이터를 보조기억장치에 저장하고 그중의
일부 데이터를 번갈아 가면서 주기억장치로
적재해서 정렬하는 방식

기본 개념


내부 정렬
정렬 방식
비교 기반
분포 기반
선택 selection 정렬
버블 bubble 정렬
삽입 insertion 정렬
O(n²)
퀵 quick 정렬
합병 merge 정렬
O(nlogn)
계수 counting 정렬
기수 radix 정렬
O(n)

기본 개념

정렬을 위한 기본 가정



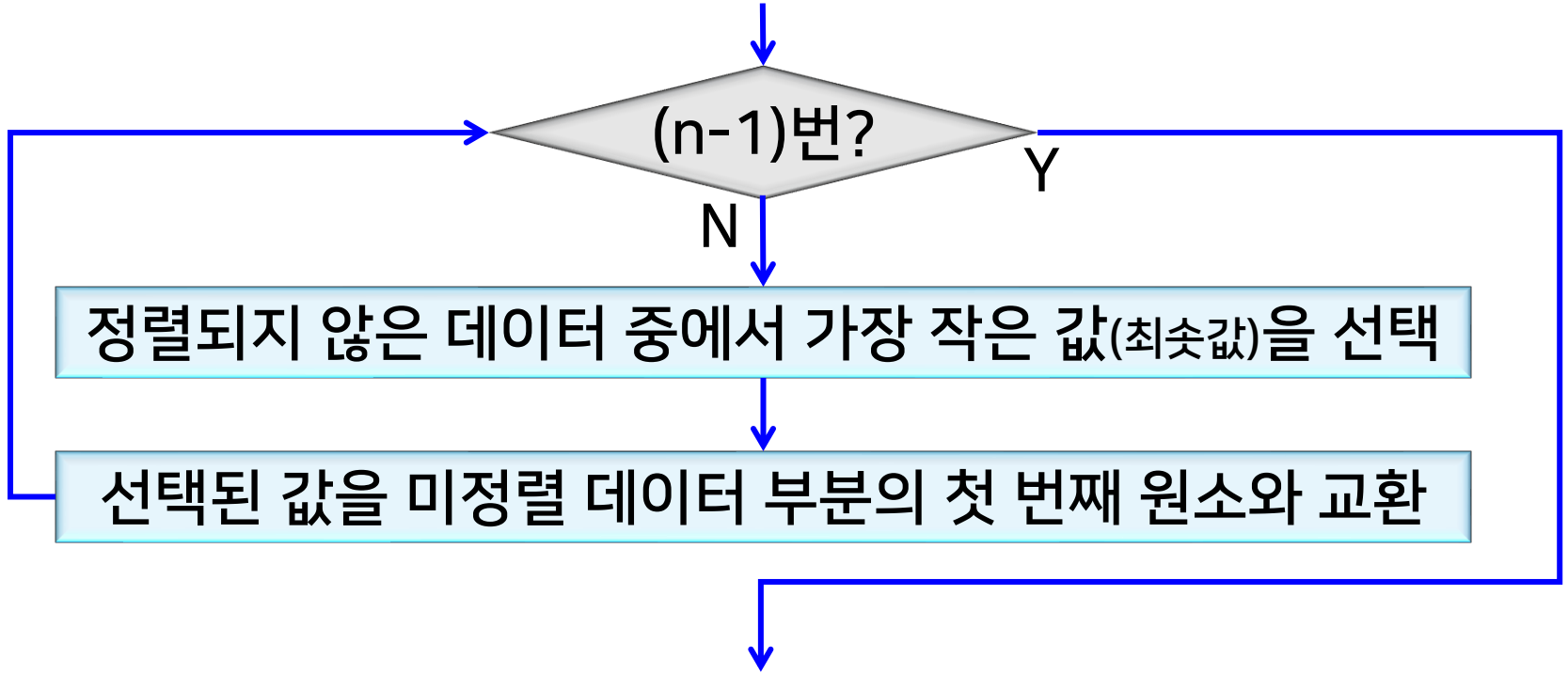
A[i] > 0, (0≤i≤n-1)

if (i < j) then A[i]≤A[j], (0≤i,j≤n-1)

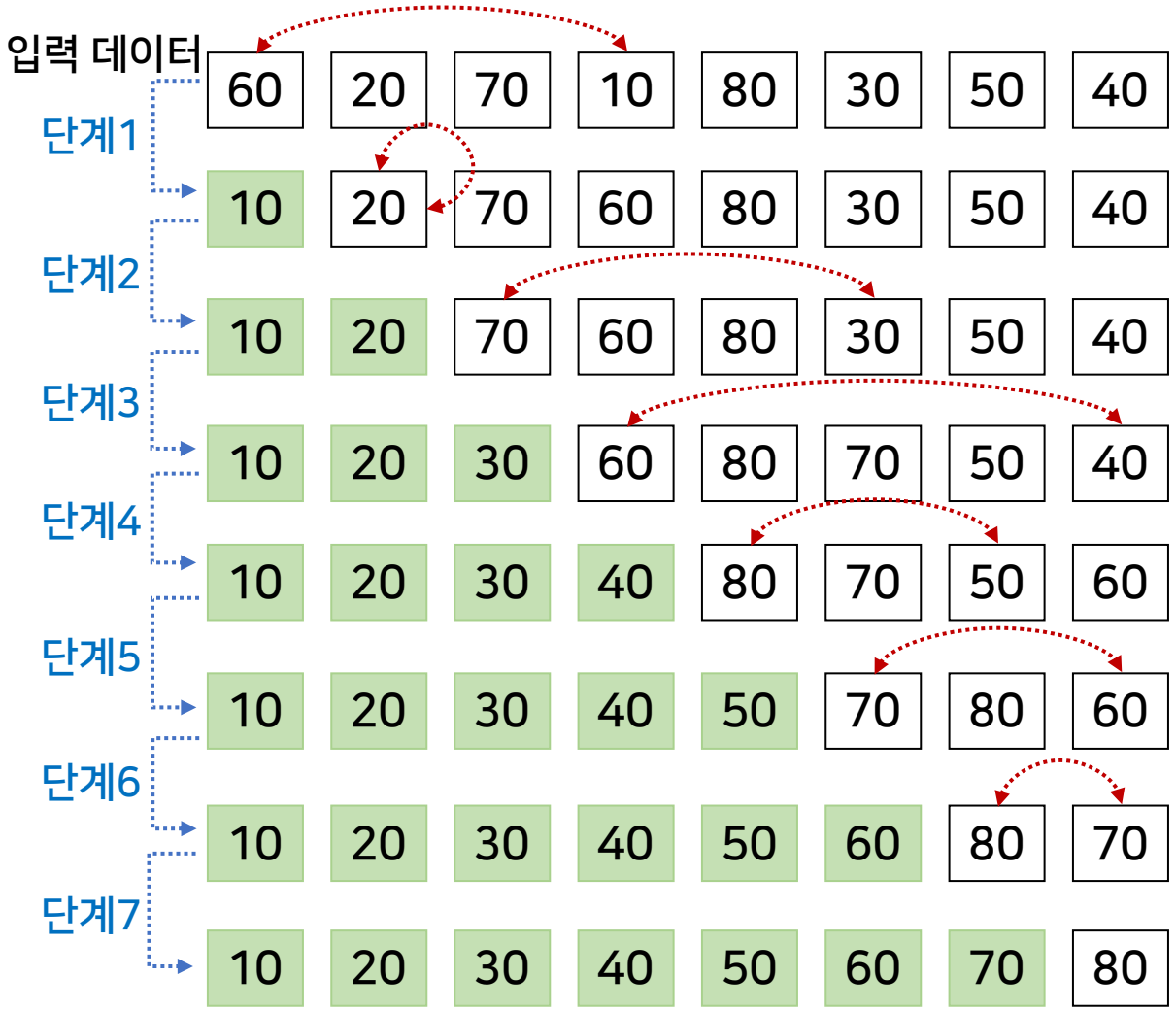
- 키의 개수 → n
- 키 저장 → A[0..n-1]
- 키값 → 양의 정수
- 정렬 방식 → 오름차순

선택 정렬

- 주어진 데이터 중에서 **가장 작은 값부터** 차례대로 **선택**해서 나열하는 방식



선택 정렬

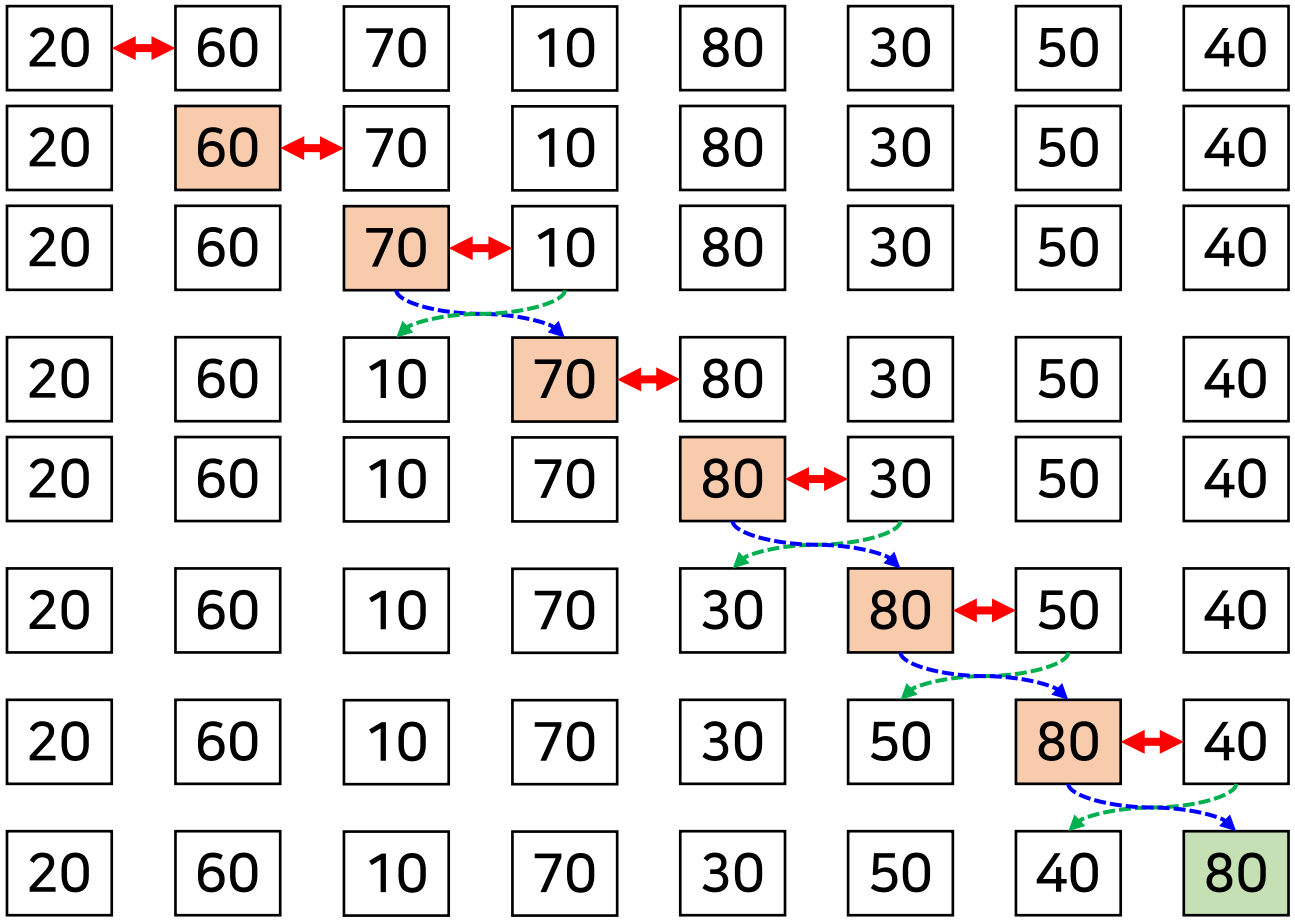


$O(n^2)$

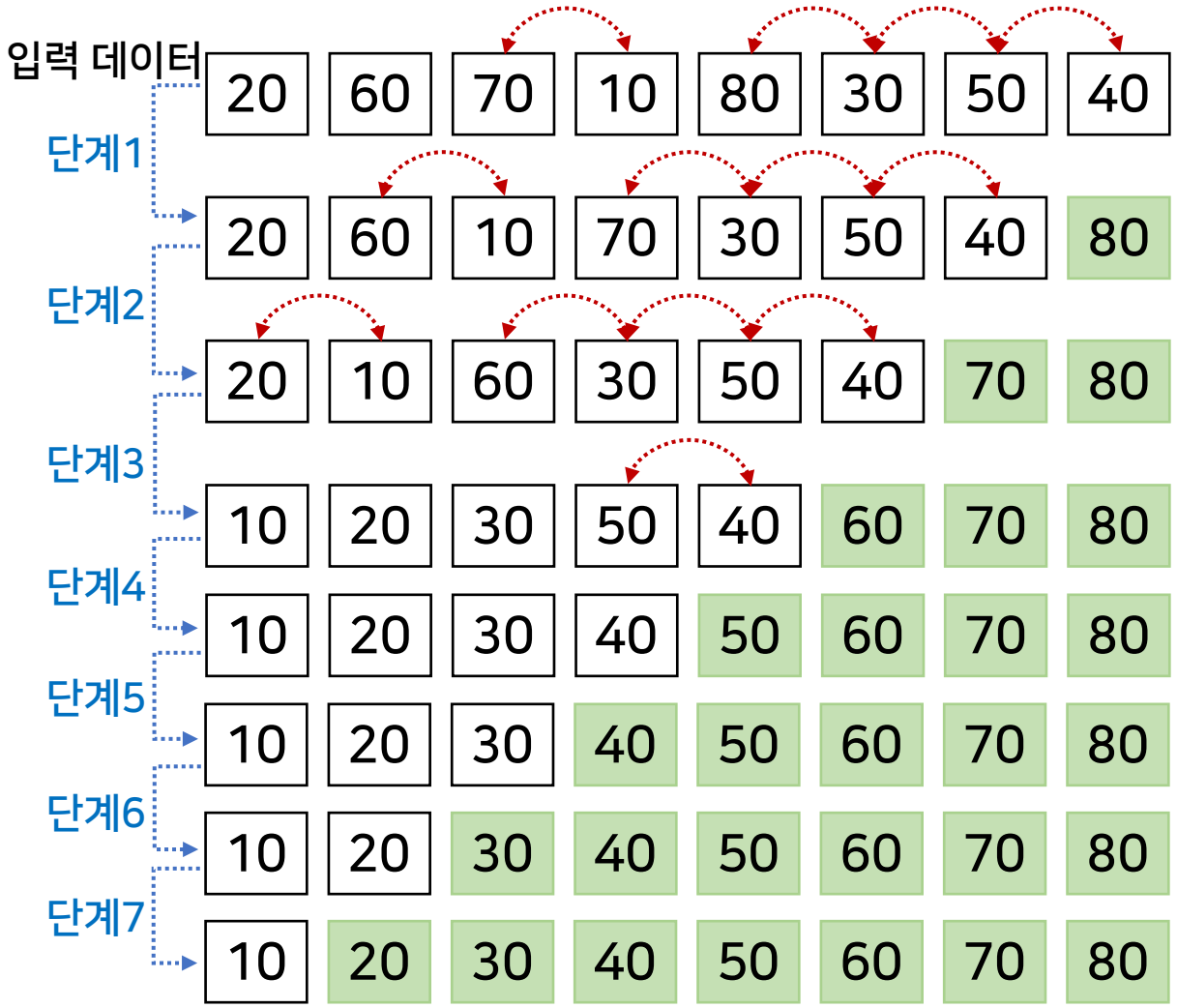
언제나 동일한 수행 시간
→ 데이터의 입력 상태에
민감하지 않은 수행 시간

버블 정렬

- 왼쪽에서부터 모든 인접한 두 데이터를 차례대로 비교하여 왼쪽의 값이 더 큰 경우에는 오른쪽 값과 자리를 바꾸는 과정을 반복



버블 정렬


입력 데이터
20 60 70 10 80 30 50 40
단계1
20 60 10 70 30 50 40 80
단계2
20 10 60 30 50 40 70 80
단계3
10 20 30 50 40 60 70 80
단계4
10 20 30 40 50 60 70 80
단계5
10 20 30 40 50 60 70 80
단계6
10 20 30 40 50 60 70 80
단계7
10 20 30 40 50 60 70 80

버블 정렬의 특징

성능 → 입력 데이터의 상태에 영향을 받음

10 20 30 40 50 → 원하는 순서로 이미 정렬되어 있는 경우

→ 최선의 경우 O(n)

50 40 30 20 10 → 역순으로 정렬되어 있는 경우

→ 최악의 경우 $O(n^2)$

40 30 20 10 50
30 20 10 40 50
20 10 30 40 50
10 20 30 40 50
10 20 30 40 50

선택 정렬에 비해 데이터 교환이 더 많이 발생

→ 선택 정렬보다 비효율적

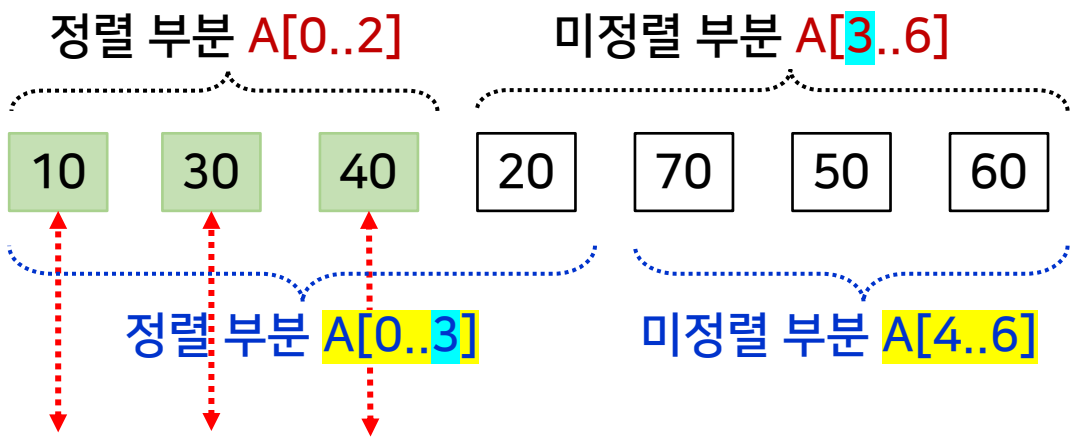
삽입 정렬

- **주어진 데이터를 하나씩 뽑은 후,**
  나열된 데이터들이 항상 정렬된 형태를 가지도록
  뽑은 데이터를 바른 위치에 삽입해서 나열하는 방식

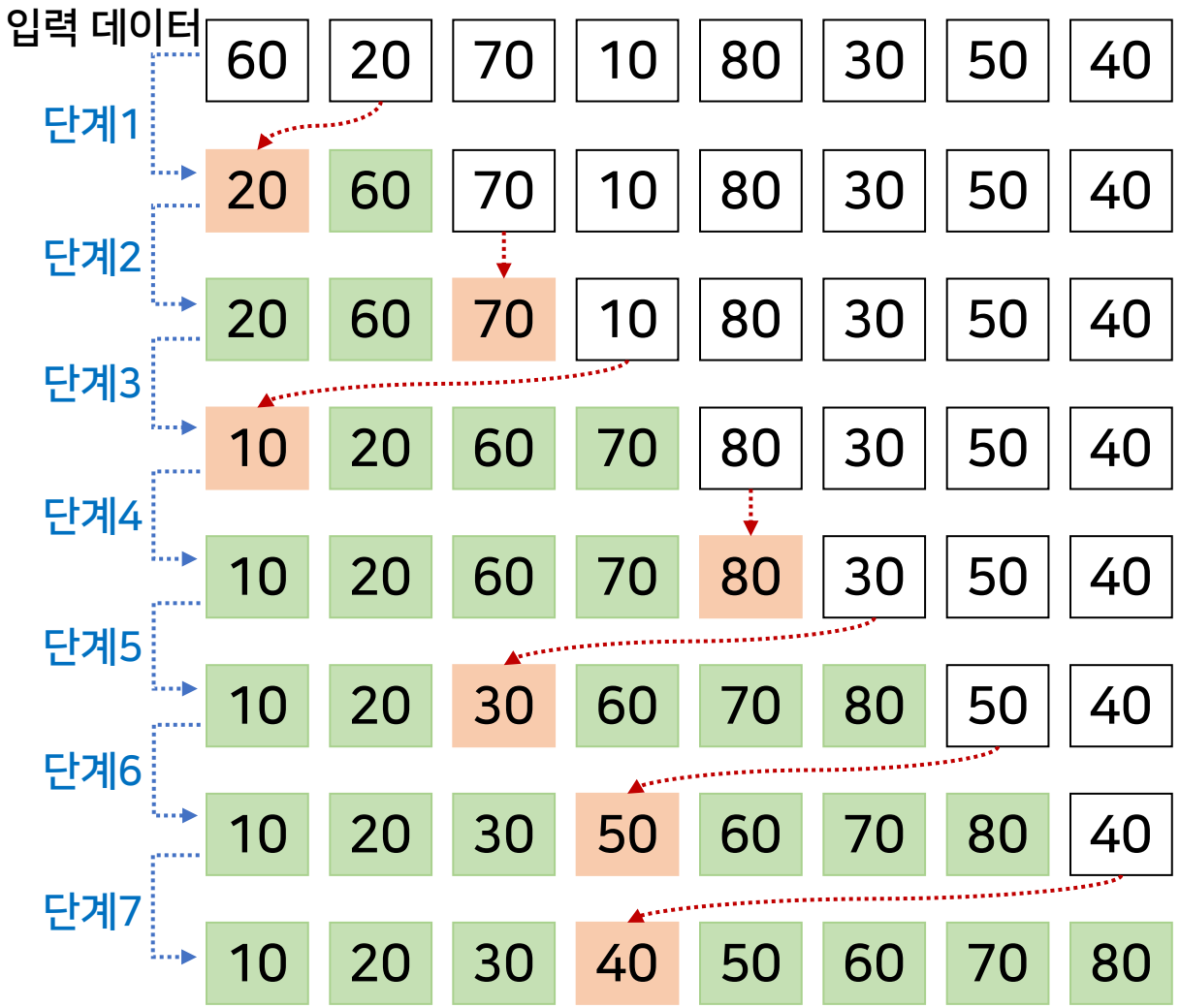
  - 입력 배열을 정렬 부분과 미정렬 부분으로 구분하고,
  - 미정렬 부분의 가장 왼쪽에 있는 데이터("첫 번째 데이터")를 뽑은 후,
    정렬된 부분에서 제자리를 찾아서 삽입하는 과정을 반복

삽입 정렬

뽑은 데이터를 삽입할 제자리를 찾는 방법



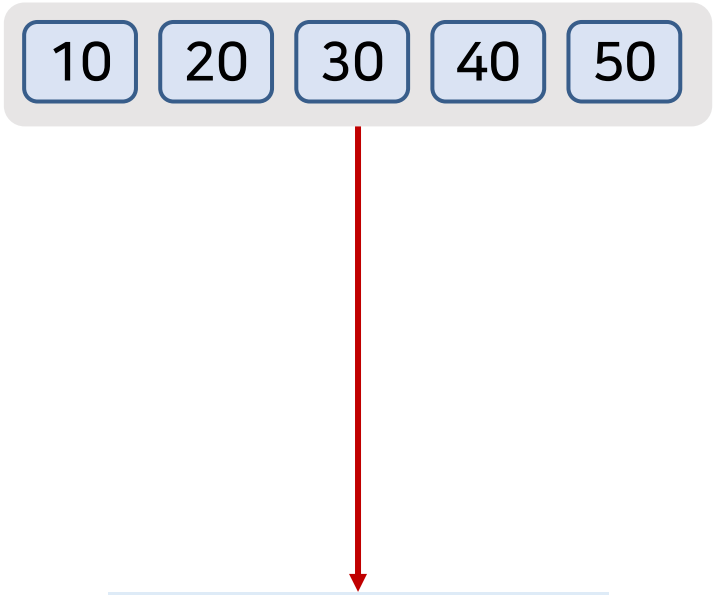
삽입 정렬


입력 데이터
60 20 70 10 80 30 50 40
단계1
20 60 70 10 80 30 50 40
단계2
20 60 70 10 80 30 50 40
단계3
10 20 60 70 80 30 50 40
단계4
10 20 60 70 80 30 50 40
단계5
10 20 30 60 70 80 50 40
단계6
10 20 30 50 60 70 80 40
단계7
10 20 30 40 50 60 70 80

삽입 정렬의 특징

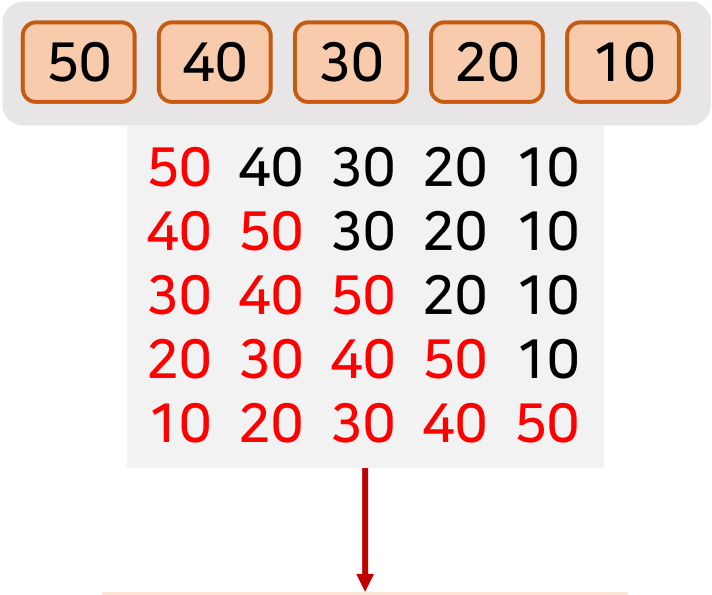
성능 → 입력 데이터의 상태에 영향을 받음

이미 정렬되어 있는 경우



최선의 경우 O(n)

역순으로 정렬된 경우



최악의 경우 $O(n^2)$

정리하기

1. 알고리즘의 개념

- 조건 → 입출력, 명확성, 유한성, 유효성 + (효율성)
- 생성단계: 설계 → 표현/기술 → 정확성분석 → 효율성분석

2. 알고리즘의 설계

- 분할정복방법, 동적 프로그래밍방법, 욕심쟁이방법(거스름돈 문제, 배낭 문제)

3. 알고리즘의 분석

- 정확성분석 + 효율성분석(→ 공간 복잡도, 시간 복잡도)
- 시간 복잡도 → 알고리즘의 수행시간(입력 크기의 함수, 최악 수행시간)
- 점근성능 → 개념, 표기법(O, Ω, Θ), O-표기의 크기관계

4. 정렬 알고리즘: 선택 정렬, 버블 정렬, 삽입 정렬

- 내부정렬, 비교기반
- 선택정렬 → $O(n^2)$, 언제나 동일한 수행시간
- 버블정렬, 삽입정렬 → $O(n^2)$, 입력데이터의 상태에 민감(최악 $O(n^2)$, 최선 $O(n)$)

06 강

다음시간 안내

알고리즘 (2)