01

알고리즘의 개요

01. 알고리즘의 개요

I. 알고리즘의 개념

- **알고리즘(Algorithm)** : 문제를 해결하기 위한 일련의 절차로, 어떤 작업을 수행하기 위한 명령어를 입력받아 결과를 출력해 내는 과정을 기술한 것.

그림 7-1 알고리즘의 예시 : 라면 끓이기

01. 알고리즘의 개요

II. 알고리즘의 표현 방법

▪ 자연어

- **자연어(Natural Language) :** 사람이 사용하는 자연 언어.
- 자연어로 표현된 알고리즘은 사람에게 친숙한 언어로 구성되어 있어 쉽게 작성할 수 있지만, 명확하지 않은 표현 때문에 오해가 발생할 수 있음.

```
시작
    a에 4를 대입한다.
    b에 2를 대입한다.
    a와 b를 더한 값을 c에 넣는다.
    c 값을 출력한다.
끝
```

그림 7-2 자연어로 표현한 알고리즘의 예

01. 알고리즘의 개요

II. 알고리즘의 표현 방법

▪ 순서도

- **순서도(Flow Chart) :** 기호와 선을 이용해 문제 해결에 필요한 처리 과정을 표현한 방법.
- 순서도는 전체적인 구조 흐름을 파악하는 데 적합하지만, 복잡한 알고리즘을 표현할 때는 그림도 복잡해져 효과가 떨어짐.

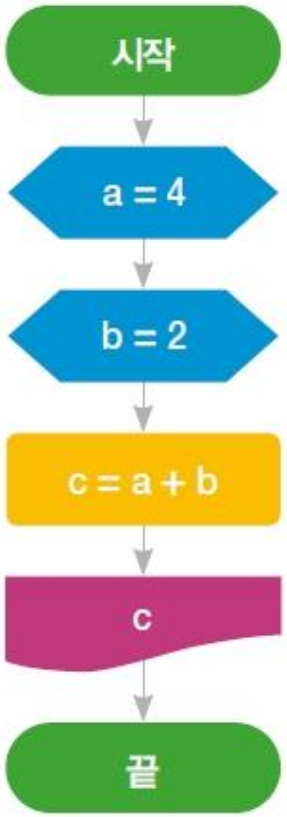


그림 7-3 순서도로 표현한 알고리즘의 예

II. 알고리즘의 표현 방법

▪ 의사코드

- **의사코드(Pseudo Code) :** 프로그래밍 언어로 작성된 코드와 비슷한 형태로 알고리즘을 표현하는 방법으로, 구조는 프로그래밍 언어와 비슷하지만 특정 프로그래밍 언어의 문법을 따르지는 않음.

```
START
    a = 4
    b = 2
    c = a + b
    PRINT c
END
```

그림 7-4 의사코드로 표현한 알고리즘의 예

01. 알고리즘의 개요

II. 알고리즘의 표현 방법

▪ 프로그래밍 언어

- **프로그래밍 언어(Programming Language) :** 프로그래밍을 할 때 사용하는 언어로, 컴퓨터가 이해할 수 있는 언어임.

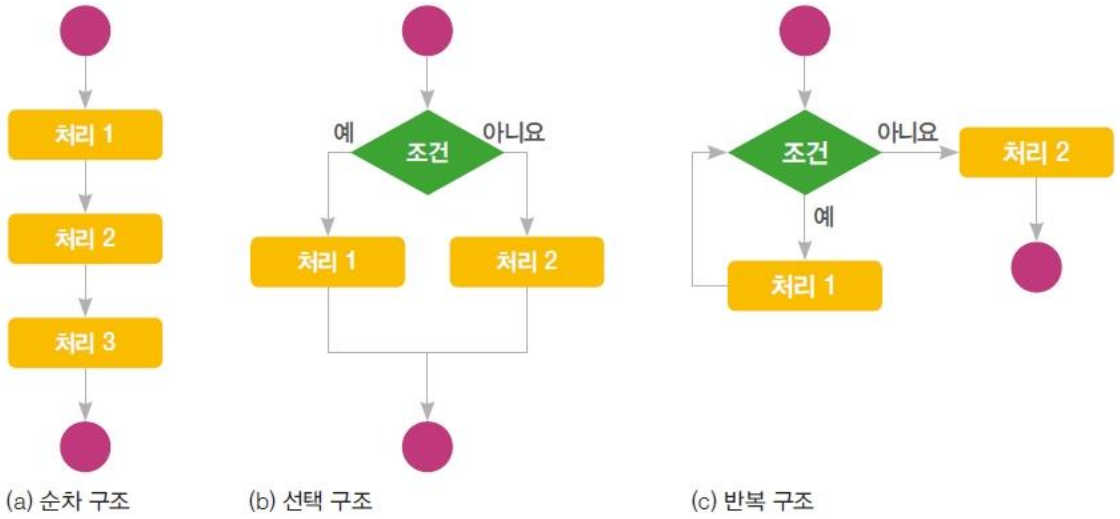
```
>>> a = 4
>>> b = 2
>>> c = a + b
>>> c
6
```

그림 7-5 프로그래밍 언어로 표현한 알고리즘의 예

01. 알고리즘의 개요

III. 알고리즘의 설계

- **알고리즘의 설계** : 가장 효율적인 문제 해결 방법을 찾아내는 과정.
- **알고리즘 제어 구조의 종류**
  - 순차 구조 : 명령을 순서대로 하나씩 수행하는 구조.
  - 선택 구조 : 조건의 결과에 따라 명령을 선택해서 실행하는 구조.
  - 반복 구조 : 같은 동작을 여러 번 반복해 수행해야 할 때 실행하는 구조.



(a) 순차 구조 (b) 선택 구조 (c) 반복 구조

그림 7-6 알고리즘의 제어 구조

01. 알고리즘의 개요

III. 알고리즘의 설계

하나 더 알기 **알고리즘의 조건**

- **입력 :** 알고리즘에 입력되는 자료가 0개 이상 존재해야 함.
- **출력 :** 알고리즘이 실행되면 결과 값이 1개 이상 나와야 함.
- **유한성 :** 알고리즘은 반드시 종료되어야 함.
- **명확성 :** 알고리즘의 명령은 모호하지 않고 명확해야 함.
- **실행 가능성 :** 알고리즘의 모든 명령은 실행 가능해야 함.

02
정렬 알고리즘

02. 정렬 알고리즘

- **정렬(Sort) :** 데이터를 일정한 규칙에 따라 재배열하는 것.
- **정렬 알고리즘(Sort Algorithm) :** 주어진 데이터를 정해진 순서대로 나열하는 알고리즘.
- **정렬 알고리즘의 종류**
  - 선택 정렬
  - 버블 정렬
  - 삽입 정렬

눈에 보이는 데이터 정렬**(04:41)**

02. 정렬 알고리즘

I. 선택 정렬

- **선택 정렬(Selection Sort) :** 아무렇게나 놓인 데이터 중 가장 작은 데이터의 위치를 가장 앞에 있는 데이터 위치와 바꾸는 알고리즘.
- **선택 정렬 진행 과정**

① 정렬되지 않은 데이터 중 최솟값을 찾는다.

② 이 최솟값과 첫 번째 자리에 있는 데이터의 위치를 서로 바꾼다.

③ 위 작업을 계속 반복한다.

02. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시  선택 정렬 과정(01:35)

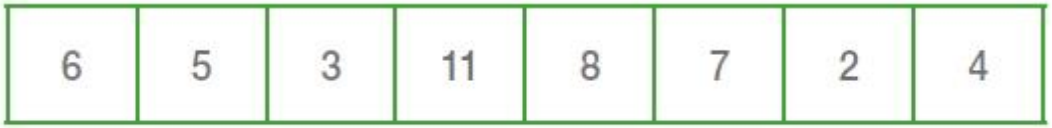


그림 7-9 정렬되지 않은 데이터들의 집합

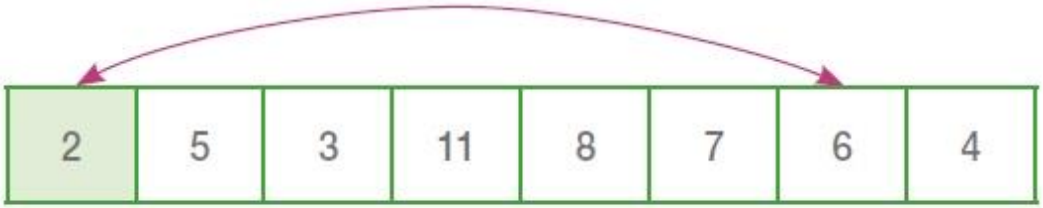


그림 7-10 선택 정렬 과정 1

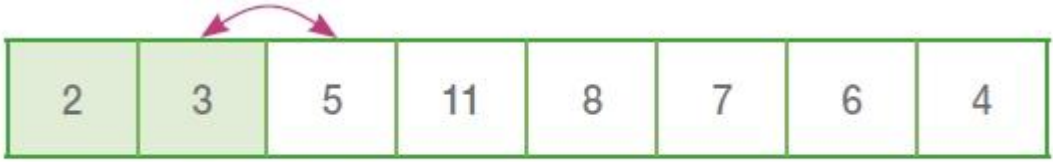


그림 7-11 선택 정렬 과정 2

02. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시

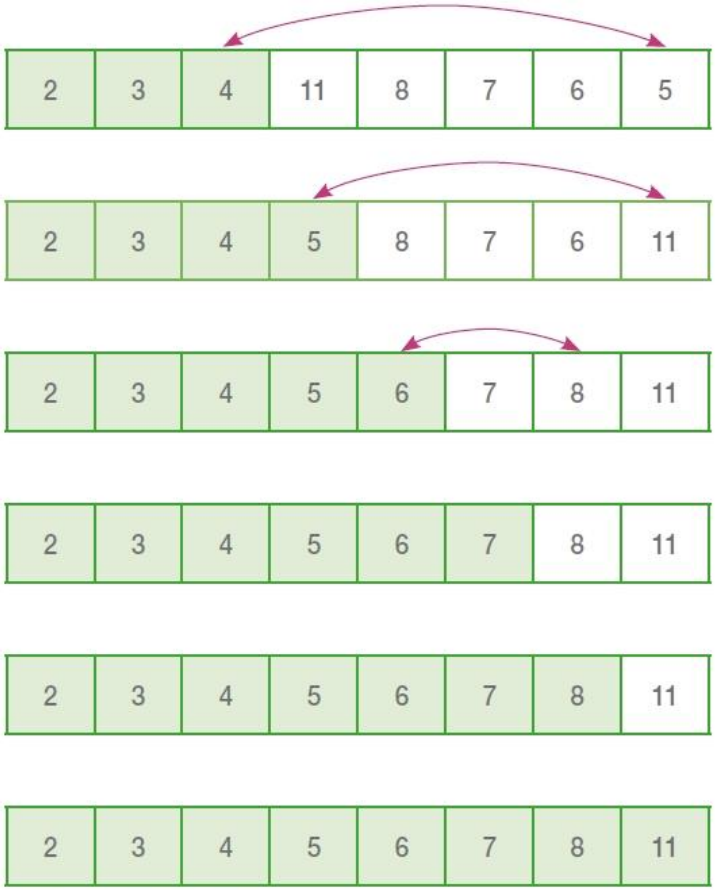


그림 7-12 선택 정렬 과정 3

02. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시

그림 7-13 선택 정렬 과정 4

02. 정렬 알고리즘

I. 선택 정렬

하나 더 알기 파이썬으로 구현하는 선택 정렬

코드 7-1 파이썬으로 구현하는 선택 정렬

```
01 import random
02
03 def my_selection_sorting(data):
04     for index in range(len(data) - 1):
05         lowest = index
06         for index2 in range(index + 1, len(data)):
07             if data[lowest] > data[index2]:
08                 lowest = index2
09         data[index], data[lowest] = data[lowest], data[index]
10     return data
11
12
13 a = random.sample(range(100), 10)
14 print('a: ', a)
15
16 b = my_selection_sorting(a)
17 print('a: ', a)
18 print('b: ', b)
19
```

출력 결과

```
a:  [88, 66, 91, 54, 22, 9, 32, 15, 17, 85]
a:  [9, 15, 17, 22, 32, 54, 66, 85, 88, 91]
b:  [9, 15, 17, 22, 32, 54, 66, 85, 88, 91]

# 출력 결과는 a 값에 따라 매번 달라진다.
```

02. 정렬 알고리즘

II. 버블 정렬

- **버블 정렬(Bubble Sort) :** 나란히 놓인 데이터 두 개 중 숫자가 큰 데이터를 뒤로 보내며 정렬하는 방식.
- 버블 정렬은 왼쪽에 놓인 데이터에서부터 시작하며, 크기순으로 자리를 서로 바꿈.

02. 정렬 알고리즘

II. 버블 정렬

- 버블 정렬 과정 예시

버블 정렬 과정(00:59)

그림 7-14 정렬되지 않은 데이터 집합

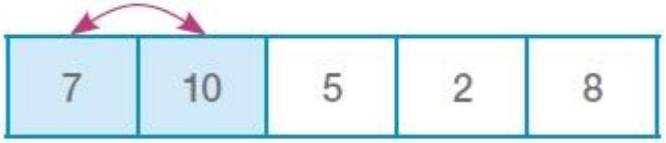


그림 7-15 버블 정렬 과정 1

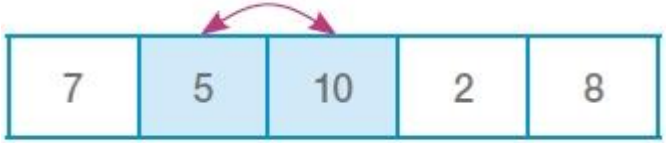


그림 7-16 버블 정렬 과정 2

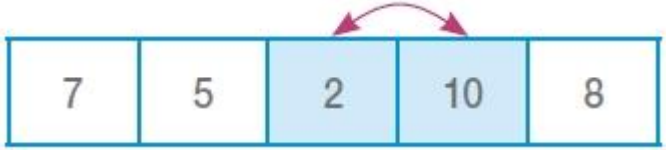


그림 7-17 버블 정렬 과정 3

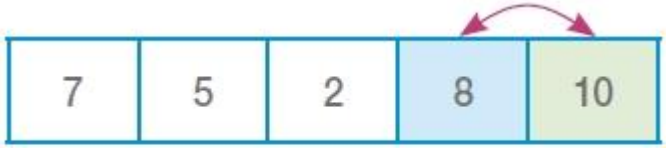


그림 7-18 버블 정렬 과정 4

02. 정렬 알고리즘

II. 버블 정렬

- 버블 정렬 과정 예시

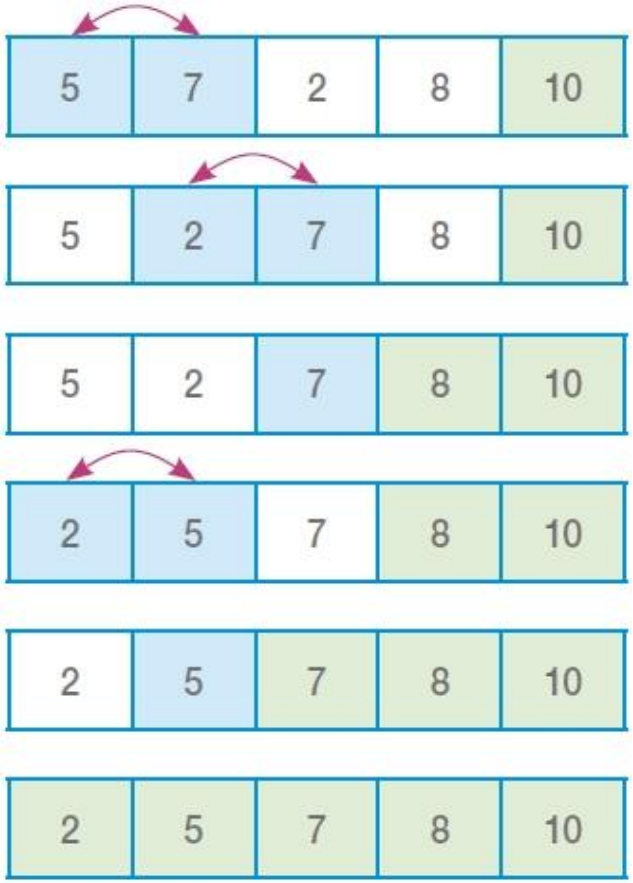


그림 7-19 버블 정렬 과정 5

02. 정렬 알고리즘

III. 삽입 정렬

- **삽입 정렬(Insertion Sort) :** 아직 정렬되지 않은 데이터를 이미 정렬된 부분 중 한 곳에 삽입해 정렬하는 방식.
- 삽입정렬은 해당 데이터가 들어갈 적절한 위치를 찾아 줌.

02. 정렬 알고리즘

III. 삽입 정렬

- **삽입 정렬 과정 예시**

삽입 정렬 과정(02:04)

그림 7-20 정렬되지 않은 데이터 집합

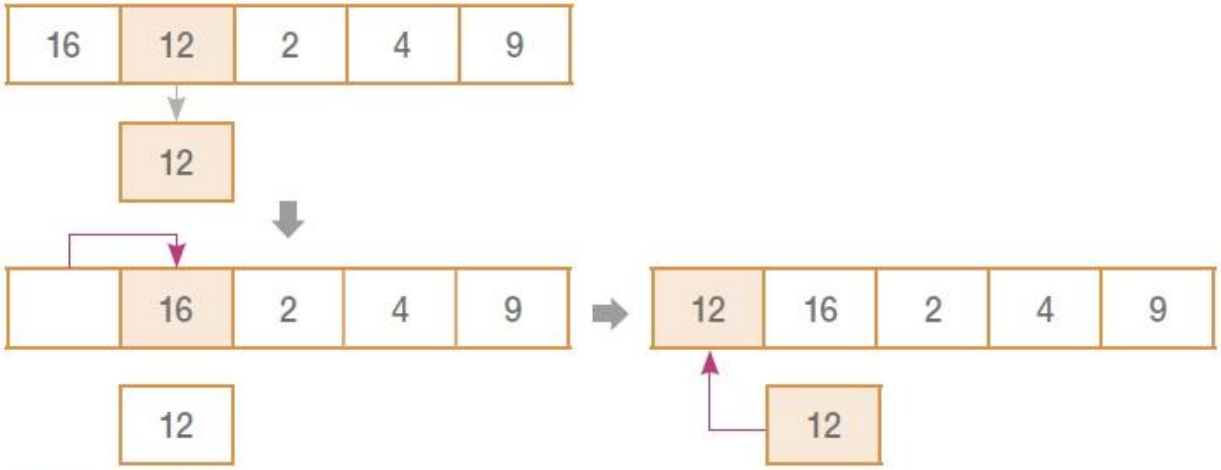


그림 7-21 삽입 정렬 과정 1

02. 정렬 알고리즘

III. 삽입 정렬

- **삽입 정렬 과정 예시**

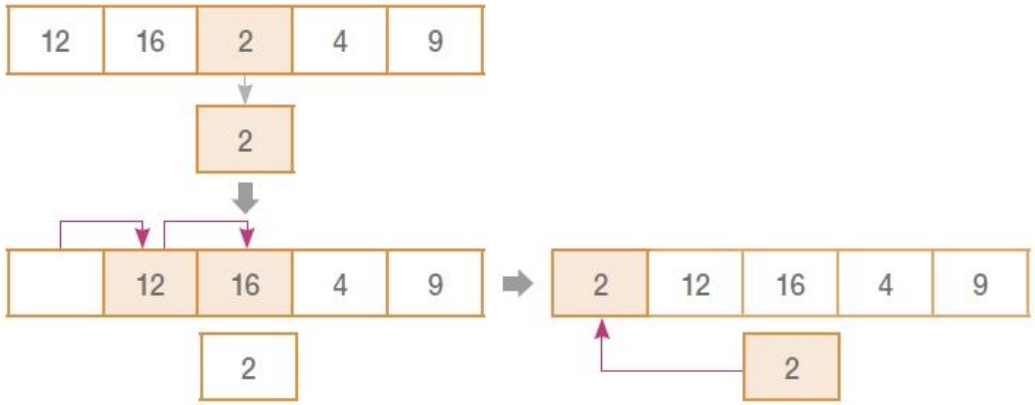


그림 7-22 삽입 정렬 과정 2

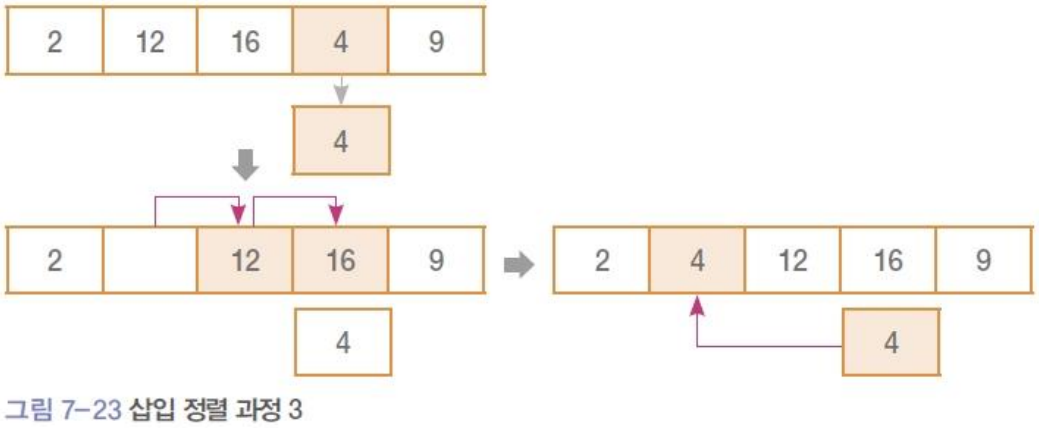


그림 7-23 삽입 정렬 과정 3

02. 정렬 알고리즘

III. 삽입 정렬

- 삽입 정렬 과정 예시

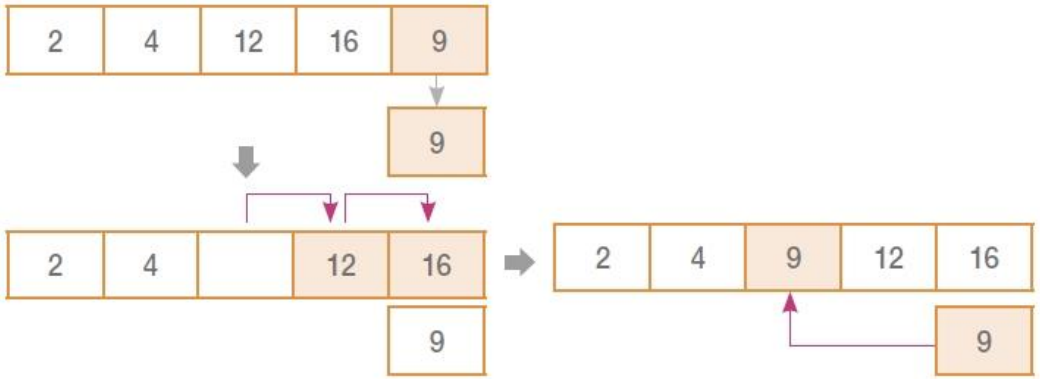


그림 7-24 삽입 정렬 과정 4

그림 7-25 삽입 정렬 과정 5

02. 정렬 알고리즘

III. 삽입 정렬

하나 더 알기 **병합 정렬**

- **병합 정렬(Merge Sort) :** 고급 정렬 알고리즘 중 하나로, 정렬된 여러 개의 자료 집합을 병합하여 하나의 정렬된 집합으로 만드는 정렬 방법.
- **병합 정렬 수행 과정**
  - 분할 : 자료들을 두 개의 부분집합으로 분할한다.
  - 정복 : 부분집합에 있는 원소를 정렬한다.
  - 정렬된 부분집합들을 하나의 집합으로 정렬하여 결합한다.

III. 삽입 정렬

하나 더 알기 **병합 정렬**

- **병합 정렬 예시(Merge Sort)**

① **분할 단계 :** 전체 자료의 집합이 최소 원소의 부분집합이 될 때까지 분할 작업을 반복한다.

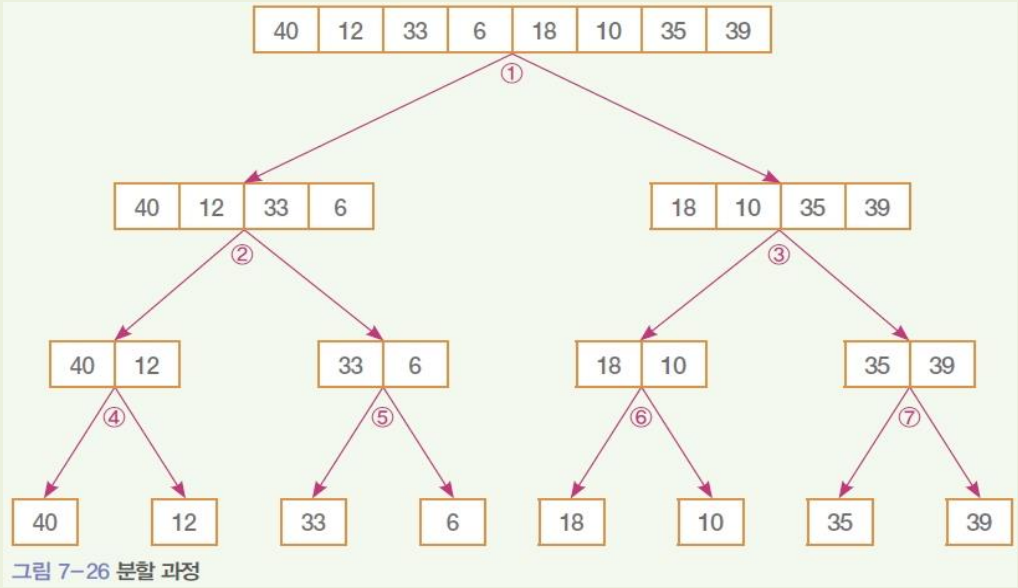


그림 7-26 분할 과정

III. 삽입 정렬

하나 더 알기 **병합 정렬**

- **병합 정렬 예시(Merge Sort)**

② **정복 및 결합 단계 :** 부분집합 두 개를 정렬하여 하나로 결합한다. 부분집합들이 다시 하나의 집합으로 묶일 때까지 이 과정을 반복한다.

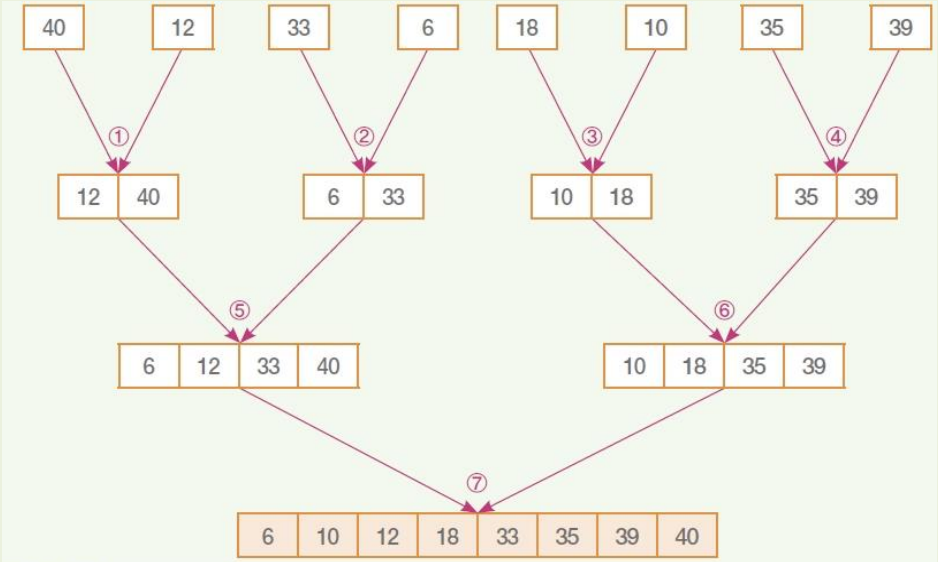


그림 7-27 정복과 결합 과정

03
검색 알고리즘

03. 검색 알고리즘

- **검색(Search) :** 특정 데이터 집합에서 어떤 조건이나 성질을 만족하는 데이터를 찾는 것.
- **검색 알고리즘(Search Algorithm) :** 검색의 개념을 활용한 알고리즘
- **검색 알고리즘의 종류 :** 순차 검색, 이진 검색, 해싱

03. 검색 알고리즘

I. 순차 검색

- **순차 검색(Sequential Search) :** 일렬로 나열된 데이터를 처음부터 끝까지 순서대로 검색하는 방법으로, 배열이나 연결 리스트로 구현된 선형 자료구조에서 자주 쓰임.
- 검색해야 하는 자료의 양에 따라 효율이 달라지며, 검색해야 할 데이터가 정렬된 상태인지 아닌지에 따라 검색 실패를 판단하는 시점도 달라짐.

03. 검색 알고리즘

I. 순차 검색

▪ 정렬되어 있지 않은 데이터의 순차 검색

순차 검색(01:45)

- 첫 번째 데이터부터 마지막 데이터까지 순서대로 비교하며 찾아야 함.
- 찾는 데이터와 일치하는 데이터를 찾으면 그 데이터가 몇 번째 자리에 있는지 확인하면 됨.
- 마지막 데이터까지 비교해도 검색 값과 일치하는 데이터가 없으면 검색에 실패한 것.

(a) 정렬되어 있지 않은 데이터의 예

(b) 검색 성공의 예 : 2 검색

(c) 검색 실패의 예 : 7 검색

그림 7-28 정렬되어 있지 않은 데이터의 순차 검색 과정

03. 검색 알고리즘

I. 순차 검색

▪ 정렬된 데이터의 순차 검색

- 중간의 데이터 값이 검색 값보다 크면 찾는 데이터가 없다는 뜻이므로, 검색 실패 여부를 바로 파악할 수 있음.

그림 7-29 정렬된 데이터의 순차 검색 과정

03. 검색 알고리즘

II. 이진 검색

- **이진 검색(Binary Search) :** 데이터 가운데에 위치한 항목을 검색 값과 비교하여 검색 값이 더 크면 오른쪽 부분을 검색하고 검색 값이 더 작으면 왼쪽 부분을 검색하는 방법.
- 정렬된 데이터에서만 사용할 수 있으며, 아주 효율적.
- 원하는 값을 찾을 때까지 검색 범위를 계속 줄여 가며 빠르게 반복 수행함.
- 검색 대상인 데이터 개수를 평균 ½씩 줄이기 때문에 데이터양이 많을 때 활용하면 매우 빠른 속도로 결과를 얻을 수 있음.

03. 검색 알고리즘

II. 이진 검색

영상 : 이진 검색(02:50)

- **이진 검색 예시 : 정렬된 데이터 집합에서 15 찾기**

그림 7-30정렬된 데이터 집합

① 검색 영역의 중간 위치를 찾은 후, 해당 위치에 있는 데이터와 찾고자 하는 데이터 값을 비교.

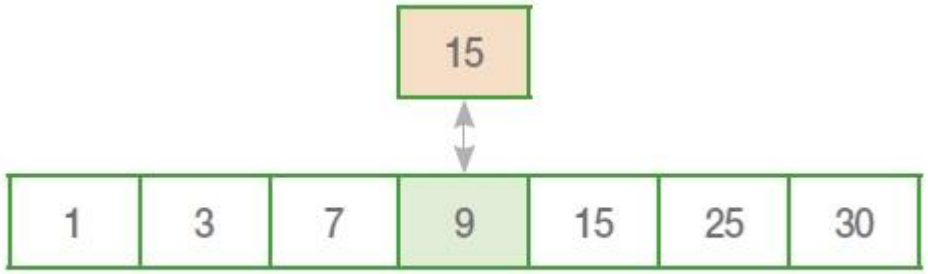


그림 7-31 이진 검색 과정 1

03. 검색 알고리즘

II. 이진 검색

- **이진 검색 예시 : 정렬된 데이터 집합에서 15 찾기**

② 데이터 15는 데이터 9보다 크므로, 찾는 데이터가 중간 위치에서 오른쪽 영역에 있음을 알 수 있다. 두 번째 검색 범위로 자리를 옮겨 중간 위치를 계산하면 여섯 번째 자리에 있는 데이터가 기준이 된다. 이 위치에 저장된 데이터 25가 데이터 15와 같은지 비교한다.

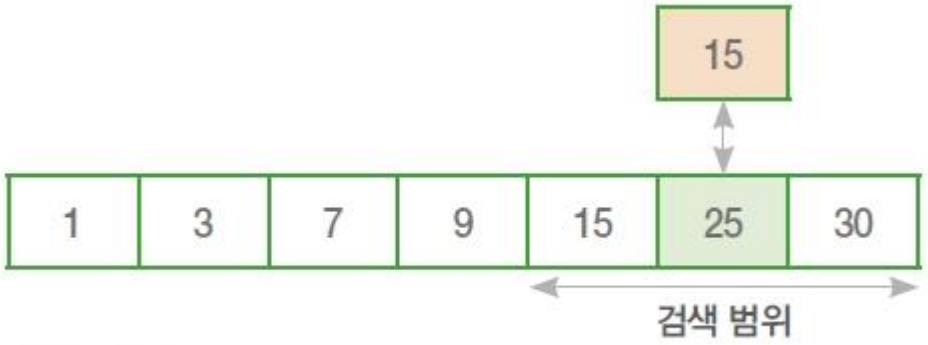


그림 7-32 이진 검색 과정 2

03. 검색 알고리즘

II. 이진 검색

- **이진 검색 예시 : 정렬된 데이터 집합에서 15 찾기**

③ 데이터 15는 데이터 25보다 작으므로 찾는 데이터는 기준에서 왼쪽 영역에 있음을 알 수 있다. 따라서 검색 범위를 다시 새로 설정해 중간 위치를 계산한다. 새로운 기준은 다섯 번째 자리의 데이터가 된다. 이 위치에 저장된 값을 비교하니 데이터 15와 같으므로 검색에 성공했다. 성공 여부를 보고한 후 종료한다.

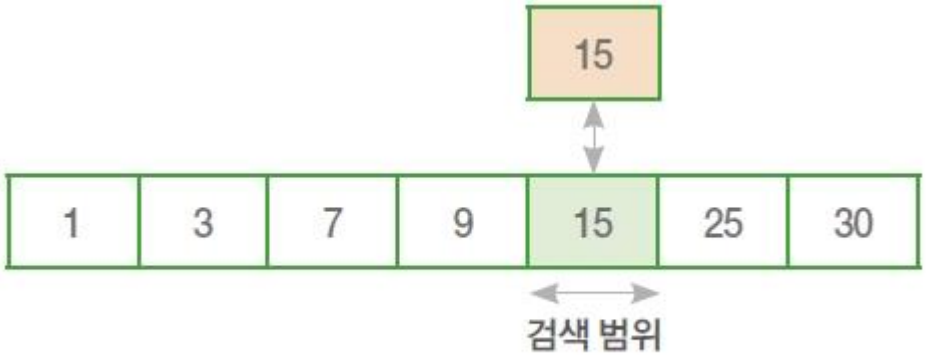


그림 7-33 이진 검색 과정 3

03. 검색 알고리즘

III. 해싱

- **해싱(Hashing) :** 산술적인 연산을 이용해 키가 있는 위치를 계산하고, 그 위치를 바로 찾아가는 방법. 검색, 삽입, 삭제 등의 연산을 효율적으로 수행함.
- **해시 함수(Hash Function) :** 키 값을 데이터 위치로 변환하는 함수.
- **해시 테이블(Hash Table) :** 해시 함수에 따라 계산된 주소 위치에 항목을 저장한 표.

그림 7-34 해싱의 검색 수행 과정

III. 해싱

하나 더 알기 **파이썬으로 구현하는 해싱**

- 파이썬에서는 딕셔너리를 이용해 해싱을 구현할 수 있음.

코드 7-2 파이썬으로 구현하는 해싱

```
01  game_keys = {'A': 'attack', 'M': 'move', 'P': 'patrol'}    # 딕셔너리 객체 생성
02
03  print(game_keys)
04
05  print(game_keys['A'])                                      # key 값으로 value 사용
06  print(game_keys['M'])
```

출력 결과

```
{'A': 'attack', 'M': 'move', 'P': 'patrol'}
attack
move
```

04
알고리즘의 분석

04. 알고리즘의 분석

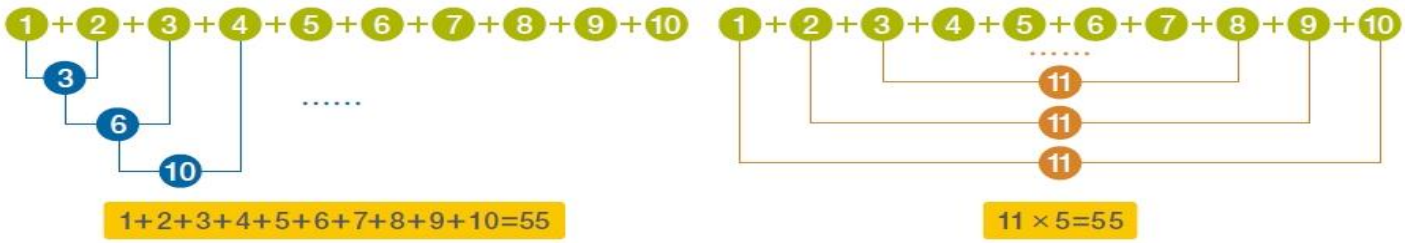
I. 알고리즘의 복잡도

- 알고리즘은 어떤 구조가 보다 효율적인지 분석해 설계해야 함
- 알고리즘이 복잡한 구조이거나 처리해야 하는 데이터가 많을 경우 알고리즘의 효율성이 프로그램 성능에 많은 영향을 끼치기 때문임
- **알고리즘의 복잡도** : 분석을 통해 알고리즘의 복잡도를 알 수 있는데 해당 알고리즘이 특정 기준에 따라 얼마나 빠르고 느리게 실행되는지를 나타낸 것.
- 'CPU의 실행 시간'은 알고리즘의 효율성을 판단하는 가장 중요한 평가 기준인데, 이 기준으로 알고리즘 효율성을 측정하는 가장 좋은 방법은 시간 복잡도를 분석하는 것.

04. 알고리즘의 분석

II. 시간 복잡도

- **시간 복잡도(Time Complexity) :** 알고리즘이 실행되고 종료될 때까지 어느 정도의 시간이 필요한지를 측정하는 방법.
  - 컴퓨터의 실행 시간을 직접 측정하기 어려우므로, 알고리즘 실행문이 실행된 횟수를 파악해 측정함
- [그림 7-7] (a)는 앞의 두 수를 더한 값을 다음 수와 차례대로 더하는 방식, (b)는 맨 앞의 수와 맨 뒤의 수를 한 쌍으로 묶어 더하고, 그 값들을 더한 횟수로 곱하는 방식.
- [그림 7-7]의 (a)와 (b)를 비교하면, '방법 2'는 '방법 1'보다 더 적은 연산 횟수로 문제를 해결하기 때문에 '방법 2'의 시간 복잡도가 '방법 1'보다 낮음.



(a) 방법 1 : 1+2+3+4+5+6+7+8+9+10=55 (b) 방법 2 : 11×5=55

그림 7-7 1부터 10까지의 수를 더하는 알고리즘 : 시간 복잡도

04. 알고리즘의 분석

III. 빅오 표기법

- **빅오 표기법(Big O Notation) :** 알고리즘의 시간 복잡도를 표현하는 방법.
- 컴퓨터에서는 입력된 값의 크기에 따라 알고리즘 처리 횟수가 얼마나 증가하는지 나타낼 때 사용.
- 입력된 데이터의 개수에 따른 알고리즘의 효율성을 분석할 때, 빅오 표기법을 이용하면 알고리즘의 복잡도를 근삿값으로 나타낼 수 있음.
- O(n) 알고리즘의 시간 복잡도가 $O(n^2)$ 알고리즘보다 낮다는 것을 알 수 있음.
  - O(n) : 알고리즘의 수행 횟수가 n만큼 커진다.
  - $O(n^2)$: 알고리즘의 수행 횟수가 n^2만큼 커진다.

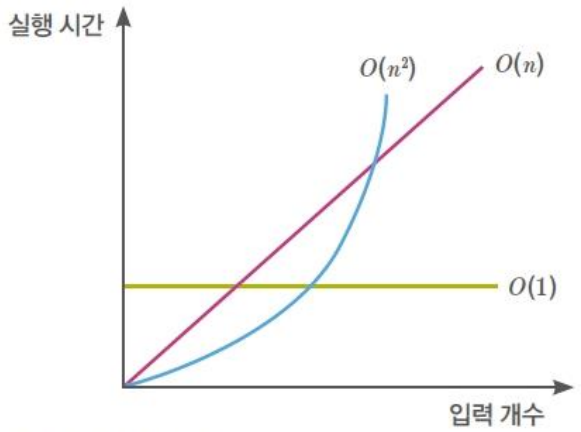


그림 7-8 빅오 표기법 그래프

04. 알고리즘의 분석

III. 빅오 표기법

하나 더 알기 **빅오 표기법 깊게 알기**

- 어떤 알고리즘의 수행 속도가 입력된 데이터의 개수 x에 따라 $f(x) = 4x^3 + 2x$ 함수로 표현된다면, 이 알고리즘의 빅오 표기법은 $O(x^3)$.
- 입력된 데이터 개수 x의 x^3에 비례하는 수행 속도를 가진다는 의미함.
- 이때 함수식에 적힌 상수 4는 별로 중요하지 않는데, 이는 상수는 효율성 분석에서 큰 비중을 차지하지 않기 때문.
- 복잡도는 입력된 데이터 개수 x 자체의 값에 제일 많은 영향을 받음.
- 동일한 이유로 2x도 별로 중요하지 않아서 빅오 표기법에서는 가장 차수가 높은 항(Term)만을 표시함.

04. 알고리즘의 분석

III. 빅오 표기법

하나 더 알기 **빅오 표기법 깊게 알기**

- **대표적인 빅오 표기법 종류**
  - O(1) : 입력 데이터의 개수와 무관하게 일정한 시간이 걸리는 경우에 사용.
  - O(log(n)) : 데이터의 양을 매번 절반으로 분할해서 처리하는 알고리즘이 해당함. 이진 검색 기법이 해당함.
  - O(n) : 알고리즘의 작업량이 입력된 데이터 개수의 상수 배. 배열의 모든 값을 출력하는 알고리즘이 해당함.
  - O(n*log(n)) : 대부분의 정렬 알고리즘은 여기에 해당함.
  - $O(n^2)$: 선택 정렬과 같은 단순한 정렬 알고리즘 대부분이 여기에 해당함.

04. 알고리즘의 분석

III. 빅오 표기법

하나 더 알기 빅오 표기법 깊게 알기

표 7-1 입력된 데이터 개수에 따른 시간 복잡도 증가율 비교

n	$log(n)$	$n * log(n)$	n^2
1	0	1	1
2	1	2	4
4	2	8	16
8	3	24	64
16	4	64	256
32	5	160	1024
64	6	384	4096
128	7	896	16384