

컴퓨터학개론

7주차 (알고리즘)

최종석(jschoi@ssu.ac.kr)



01

알고리즘의 개요

02. 알고리즘의 분석

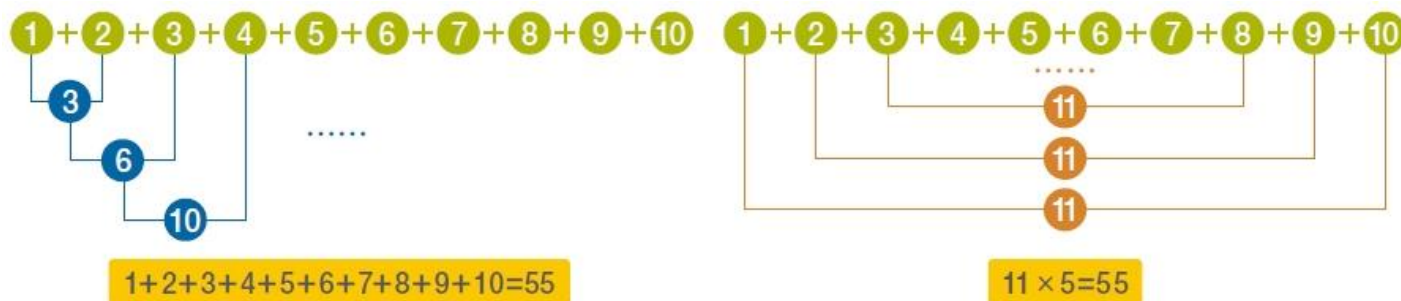
I. 알고리즘의 복잡도

- **알고리즘의 복잡도** : 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.

02. 알고리즘의 분석

II. 시간 복잡도

- **시간 복잡도(Time Complexity)** : 알고리즘이 실행되고 종료될 때까지 어느 정도의 시간이 필요한지를 측정하는 방법.
- [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 : $11 \times 5 = 55$

그림 7-7 1부터 10까지의 수를 더하는 알고리즘 : 시간 복잡도

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

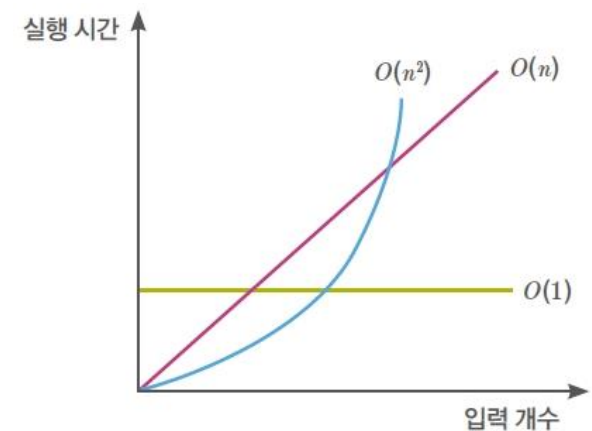


그림 7-8 빅오 표기법 그래프

02. 알고리즘의 분석

III. 빅오 표기법

하나 더 알기

빅오 표기법 깊게 알기

- 어떤 알고리즘의 수행 속도가 입력된 데이터의 개수 x 에 따라 $f(x) = 4x^3 + 2x$ 함수로 표현된다면, 이 알고리즘의 빅오 표기법은 $O(x^3)$.
- 입력된 데이터 개수 x 의 x^3 에 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 $2x$ 도 별로 중요하지 않아서 빅오 표기법에서는 가장 차수가 높은 항(Term)만을 표시함.
- 최악을 기준으로 표기됨

02. 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.
- $O(n)$: 알고리즘의 작업량이 입력된 데이터 개수의 상수 배. 배열의 모든 값을 출력하는 알고리즘이 해당함.
- $O(n \cdot \log(n))$: 대부분의 정렬 알고리즘은 여기에 해당함.
- $O(n^2)$: 선택 정렬과 같은 단순한 정렬 알고리즘 대부분이 여기에 해당함.

02. 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

03

정렬 알고리즘

03. 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

03. 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.

03. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시(Lowest 찾기)

6	5	3	11	8	7	2	4
---	---	---	----	---	---	---	---

그림 7-9 정렬되지 않은 데이터들의 집합

2	5	3	11	8	7	6	4
---	---	---	----	---	---	---	---

그림 7-10 선택 정렬 과정 1

2	3	5	11	8	7	6	4
---	---	---	----	---	---	---	---

그림 7-11 선택 정렬 과정 2

03. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시

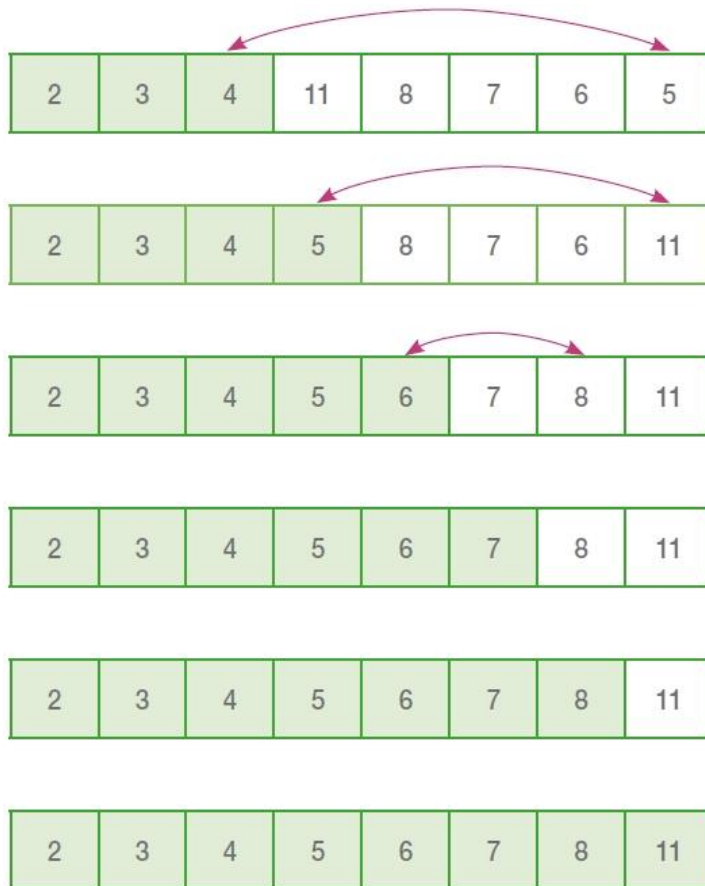
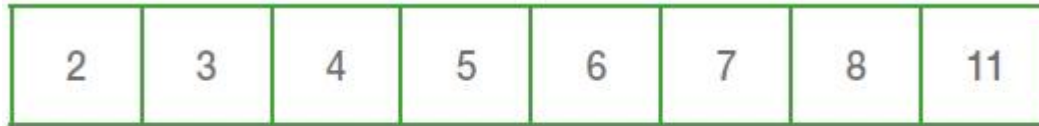


그림 7-12 선택 정렬 과정 3

03. 정렬 알고리즘

I. 선택 정렬

- 선택 정렬 과정 예시



2	3	4	5	6	7	8	11
---	---	---	---	---	---	---	----

그림 7-13 선택 정렬 과정 4

- 데이터가 n 개일 때, 1회전시 $n-1$ 의 비교 횟수가 발생
- 2회전 시 $n-2$ 의 비교 횟수가 발생.... $(n-1) + (n-2) + \dots + 2 + 1 \Rightarrow n(n-1) / 2$
- $O(n^2)$

03. 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.

03. 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.

03. 정렬 알고리즘

II. 버블 정렬

- 버블 정렬 과정 예시



그림 7-14 정렬되지 않은 데이터 집합

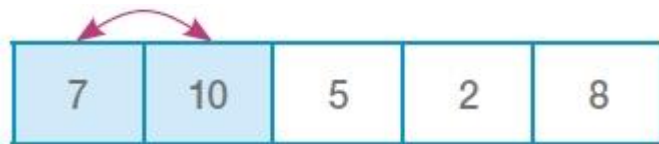


그림 7-15 버블 정렬 과정 1

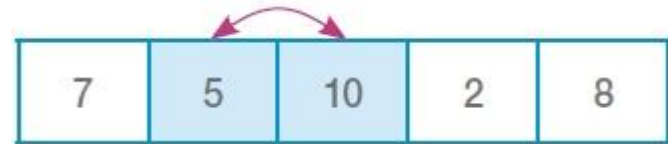


그림 7-16 버블 정렬 과정 2

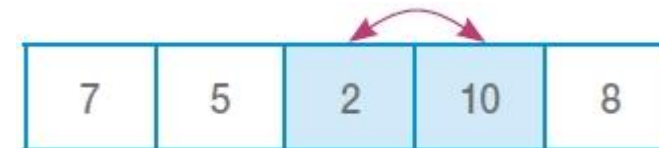


그림 7-17 버블 정렬 과정 3



그림 7-18 버블 정렬 과정 4

03. 정렬 알고리즘

II. 버블 정렬

- 버블 정렬 과정 예시

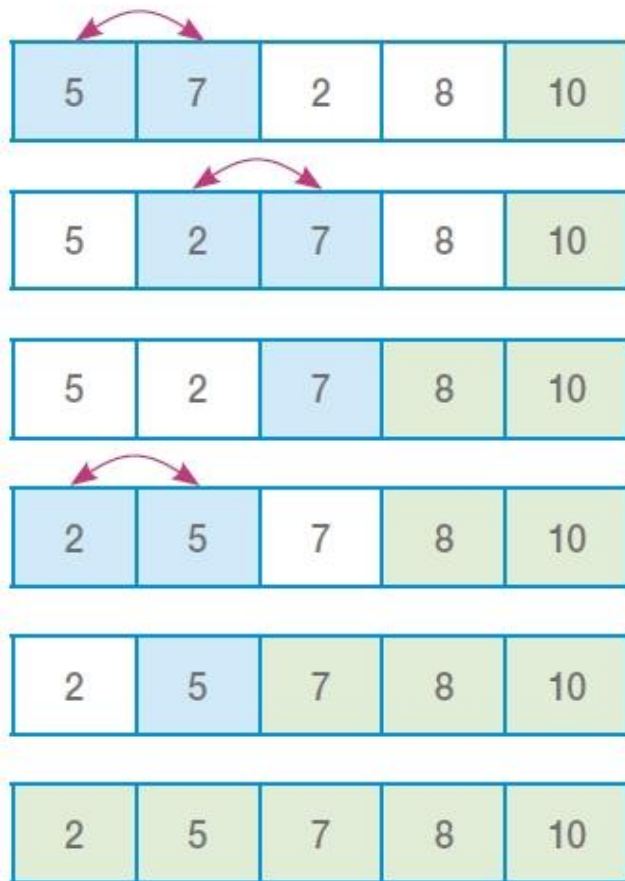


그림 7-19 버블 정렬 과정 5

03. 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.
- 첫 정렬의 시작은 2번째 인덱스부터 시작

03. 정렬 알고리즘

III. 삽입 정렬

- 삽입 정렬 과정 예시

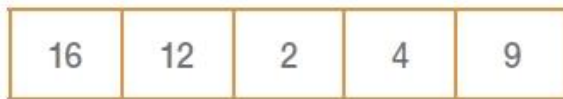


그림 7-20 정렬되지 않은 데이터 집합

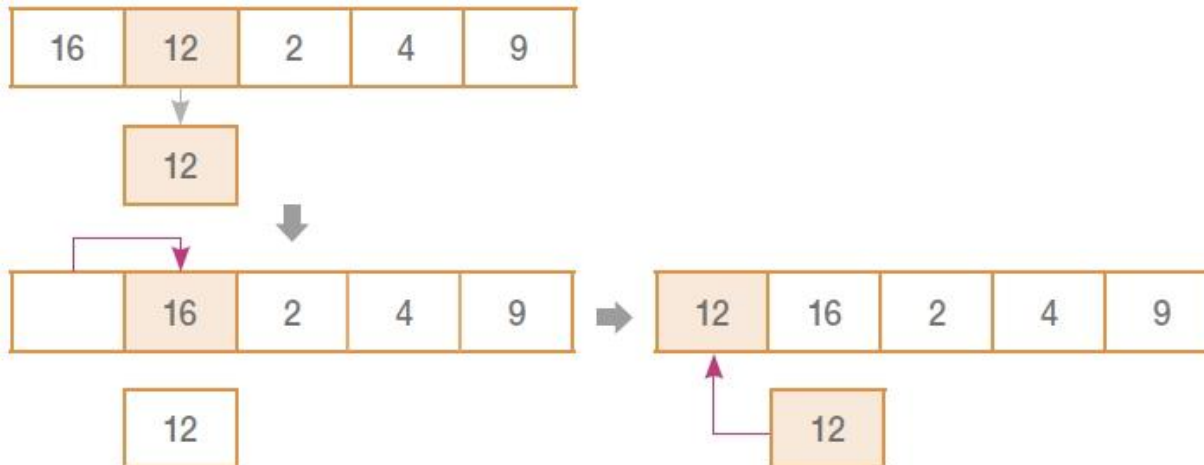


그림 7-21 삽입 정렬 과정 1

03. 정렬 알고리즘

III. 삽입 정렬

- 삽입 정렬 과정 예시

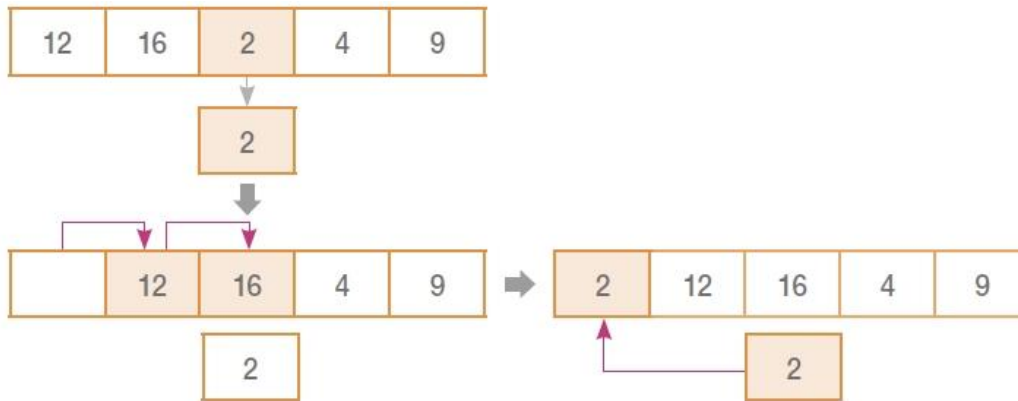


그림 7-22 삽입 정렬 과정 2

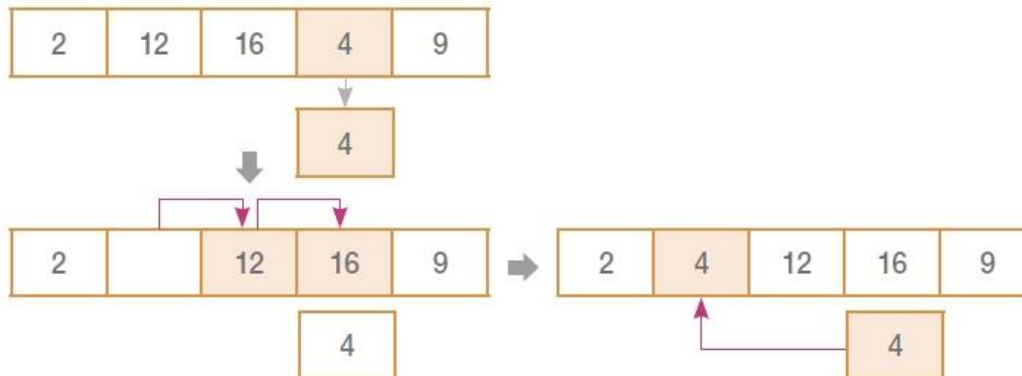


그림 7-23 삽입 정렬 과정 3

03. 정렬 알고리즘

III. 삽입 정렬

- 삽입 정렬 과정 예시

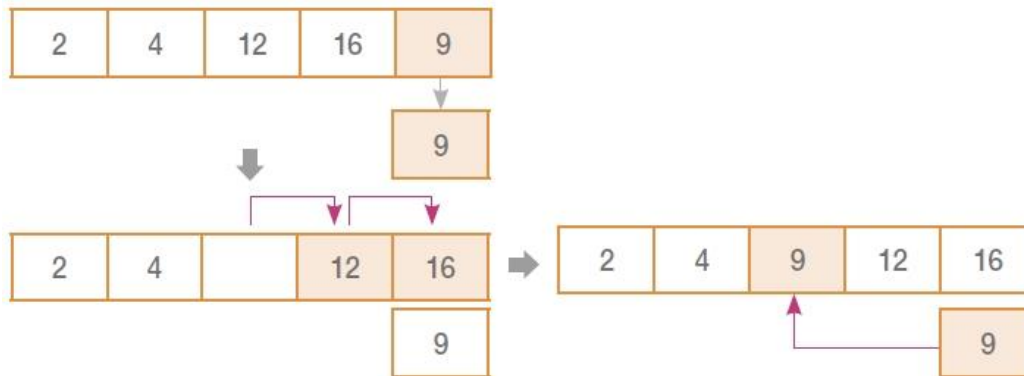


그림 7-24 삽입 정렬 과정 4



그림 7-25 삽입 정렬 과정 5

- $O(n^2)$ <- 최악의 경우(역정렬 되어있을 경우)

03. 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.

03. 정렬 알고리즘

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.

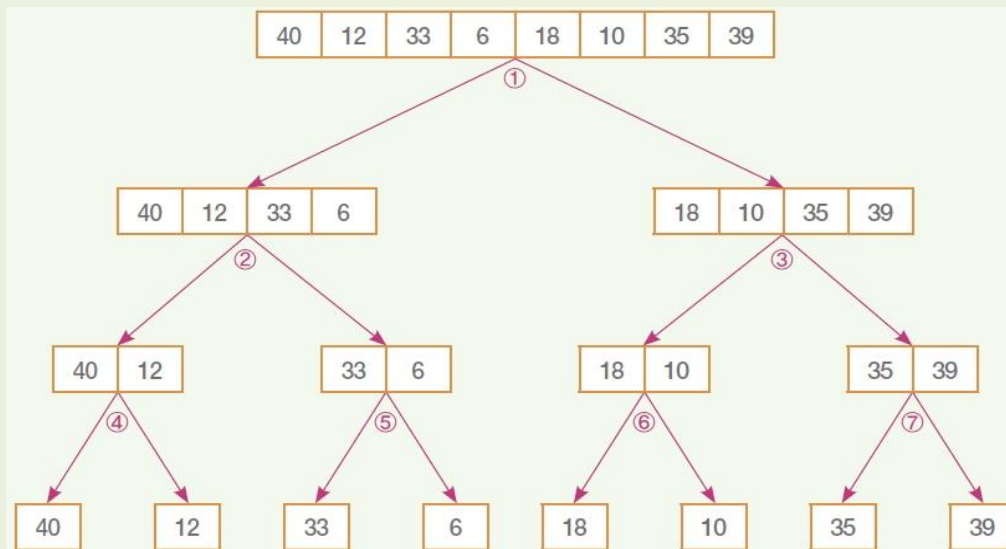


그림 7-26 분할 과정

03. 정렬 알고리즘

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.

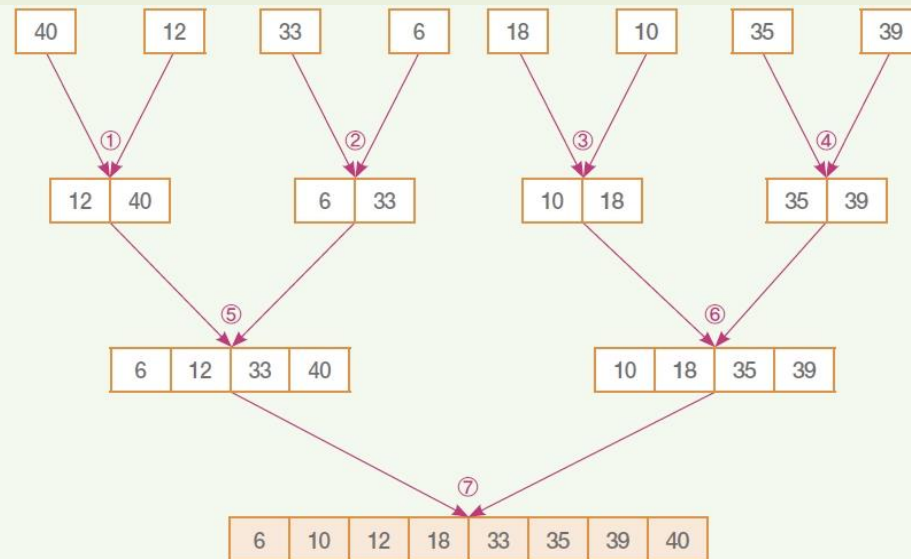


그림 7-27 정복과 결합 과정

03. 정렬 알고리즘

III. 삽입 정렬

하나 더 알기

병합 정렬

- 병합 정렬 장단점

- 단점 : 레코드를 배열로 구성하면 임시 배열이 필요하다.

레코드들의 크기가 큰 경우 이동 횟수가 많아져 매우 큰 낭비가 발생한다.

- 장점 : 데이터의 분포에 영향을 덜 받는다. 입력 데이터가 무엇이든 정렬되는 시간은 동일하다.

레코드가 연결리스트로 구성되어 있으면, 링크의 인덱스만 변경됨으로 데이터의 이동이 없다.

04

검색 알고리즘

04. 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

04. 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.

04. 검색 알고리즘

I. 순차 검색

■ 정렬되어 있지 않은 데이터의 순차 검색

- 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.

9	22	2	10	12
---	----	---	----	----

(a) 정렬되어 있지 않은 데이터의 예

① $9 \neq 2$	9	22	2	10	12
② $22 \neq 2$	9	22	2	10	12
③ $2 = 2$	9	22	2	10	12

(b) 검색 성공의 예 : 2 검색

① $9 \neq 7$	9	22	2	10	12
② $22 \neq 7$	9	22	2	10	12
③ $2 \neq 7$	9	22	2	10	12
④ $10 \neq 7$	9	22	2	10	12
⑤ $12 \neq 7$	9	22	2	10	12

(c) 검색 실패의 예 : 7 검색

그림 7-28 정렬되어 있지 않은 데이터의 순차 검색 과정

04. 검색 알고리즘

I. 순차 검색

■ 정렬된 데이터의 순차 검색

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

1	3	7	9	15
---	---	---	---	----

(a) 정렬된 데이터의 예

① $1 < 7$	1	3	7	9	15
② $3 < 7$	1	3	7	9	15
③ $7 = 7$	1	3	7	9	15

(b) 검색 성공의 예 : 7 검색

① $1 < 6$	1	3	7	9	15
② $3 < 6$	1	3	7	9	15
③ $7 > 6$	1	3	7	9	15

(c) 검색 실패의 예 : 6 검색

그림 7-29 정렬된 데이터의 순차 검색 과정

04. 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 $\frac{1}{2}$ 씩 줄이기 때문에 데이터양이 많을 때 활용하면 매우 빠른 속도로 결과를 얻을 수 있음.

04. 검색 알고리즘

II. 이진 검색

- 이진 검색 예시 : 정렬된 데이터 집합에서 15 찾기

1	3	7	9	15	25	30
---	---	---	---	----	----	----

그림 7-30 정렬된 데이터 집합

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

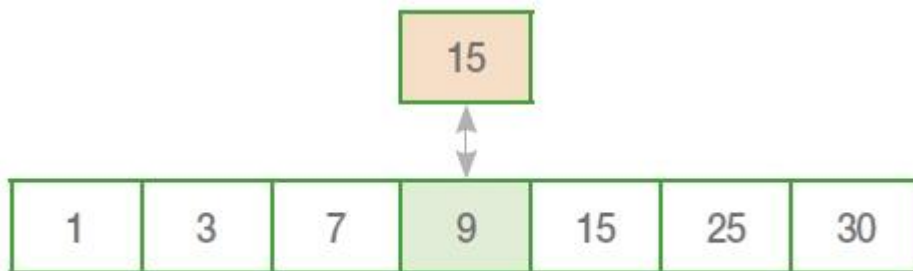


그림 7-31 이진 검색 과정 1

04. 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.

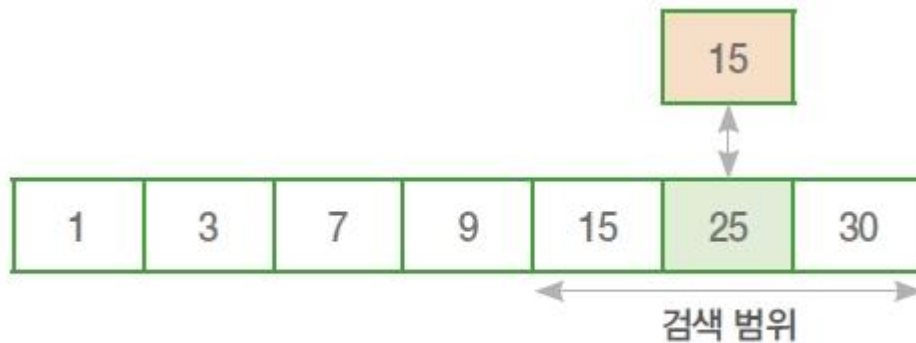


그림 7-32 이진 검색 과정 2

04. 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

04. 검색 알고리즘

III. 해싱

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

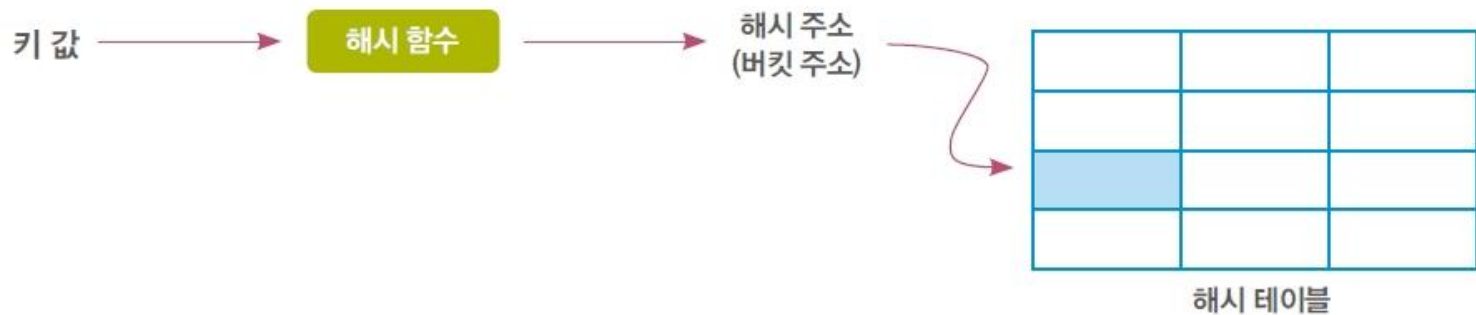


그림 7-34 해싱의 검색 수행 과정

04. 검색 알고리즘

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
```