

응용예제 01 진수 변환하기

난이도 ★ ★ ☆ ☆ ☆

예제 설명 10진수 정수를 입력하면, 2진수/8진수/16진수로 변환되어 출력되는 프로그램을 재귀 함수를 이용하여 작성한다.



실행 결과

```
Python
File Edit Shell Debug Options Window Help
===== RESTART: C:\WookData\Ex10-01.py =====
10진수 입력 --> 65535
2진수 : 1111111111111111
8진수 : 177777
16진수 : FFFF
>>> |
```

Ln: 10 Col: 4

```
## 함수 선언 부분 ##
```

```
def notation(base, n):
```

```
    if n < base :
```

```
        print(numberChar[n], end = ' ')
```

```
    else :
```

```
        notation(base, n // base)
```

```
        print(numberChar[n % base], end = ' ')
```

```
## 전역 변수 선언 부분 ##
```

```
numberChar = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
numberChar += ['A', 'B', 'C', 'D', 'E', 'F']
```

```
## 메인 코드 부분 ##
```

```
number = int(input('10진수 입력 -->'))
```

```
print("\n 2진수 : ', end = ' ')
```

```
notation(2, number)
```

```
print("\n 8진수 : ', end = ' ')
```

```
notation(8, number)
```

```
print("\n16진수 : ', end = ' ')
```

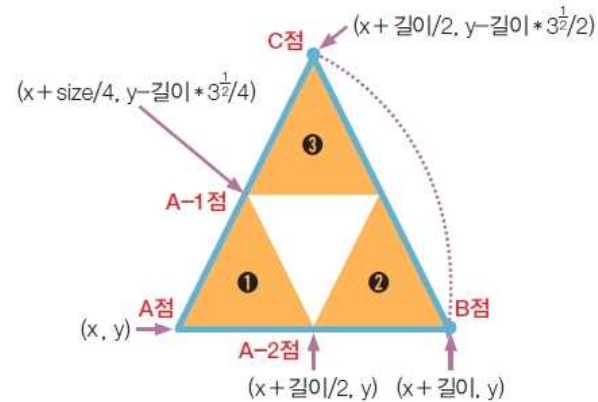
```
notation(16, number)
```

응용예제 02 시에르핀스키 삼각형 그리기

난이도★★★★☆

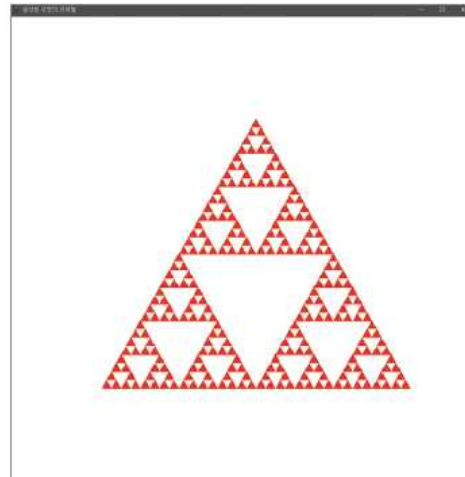
예제 설명

시에르핀스키(Sierpinski) 삼각형은 다음과 같이 큰 삼각형 안에 다시 정삼각형 3개를 균등하게 생성한다. 변의 길이와 왼쪽 아래 시작점(x, y)이 정해지면 큰 삼각형은 A점, B점, C점을 연결하면 된다. 그리고 다시 안쪽 삼각형 3개는 A점, A-1점, A-2점이 시작점이 되고 변의 길이는 1/2이 된다. 이것을 재귀 호출로 반복하면 프랙탈 모양이 된다. 각 위치를 구하는 수식은 그림에 표현했다.



Cavas.create_polygon 함수로 삼각형 사용

실행 결과



```

from tkinter import *

## 함수 선언 부분 ##
def drawTriangle(x, y, size) :
    if size >= 30 :
        drawTriangle(x, y, size/2)                # 왼쪽 아래 작은 삼각형
        drawTriangle(x+size/2, y, size / 2)        # 오른쪽 아래 작은 삼각형
        drawTriangle(x + size / 4, int(y-size*(3**0.5)/4), size / 2)  # 위쪽 작은 삼각형
    else :
        canvas.create_polygon (x, y, x + size, y, x + size / 2, y - size*(3 ** 0.5) / 2, fill = 'red', outline = "red")

## 전역 변수 선언 부분 ##
wSize = 1000
radius = 400

## 메인 코드 부분 ##
window = Tk()
window.title("삼각형 모양의 프랙탈")
canvas = Canvas(window, height = wSize, width = wSize, bg = 'white')

drawTriangle(wSize/5, wSize/5*4, wSize*2/3)

canvas.pack()
window.mainloop()

```

11

CHAPTER

정렬 기본

학습목표

- 정렬의 개념을 파악한다.
- 정렬을 위한 코드의 형식을 이해한다.
- 기본적인 정렬 방식을 학습한다.
- 정렬을 활용한 응용 프로그래밍을 작성한다.

SECTION 00 생활 속 자료구조와 알고리즘

SECTION 01 정렬의 기본

SECTION 02 기본 정렬 알고리즘의 원리와 구현

SECTION 03 기본 정렬 알고리즘의 응용

연습문제

응용예제



Section 00 생활 속 자료구조와 알고리즘

정렬이란?

학교 출석부 또는 종류에 따라 가지런히 놓여 있는 칼들처럼 순서대로 데이터가 나열되어 있는 것



Student name	Student ID	Score
1. Kim, J.	1	85
2. Lee, S.	2	78
3. Park, M.	3	92
4. Choi, H.	4	65
5. Kim, D.	5	70
6. Lee, J.	6	88
7. Park, S.	7	72
8. Choi, M.	8	80
9. Kim, H.	9	75
10. Lee, K.	10	82
11. Park, J.	11	77
12. Choi, S.	12	85
13. Kim, L.	13	70
14. Lee, P.	14	78
15. Park, K.	15	88
16. Choi, J.	16	72
17. Kim, S.	17	80
18. Lee, H.	18	75
19. Park, L.	19	82
20. Choi, K.	20	77

↑ 원하는 자리에 자유롭게 앉을 수 있는 대학교도
출석부에는 학생의 학번 순서 또는 이름 순서로
학생 명단이 작성되어 있음



↑ 가지런히 놓고 사용하면 더 편리한 칼

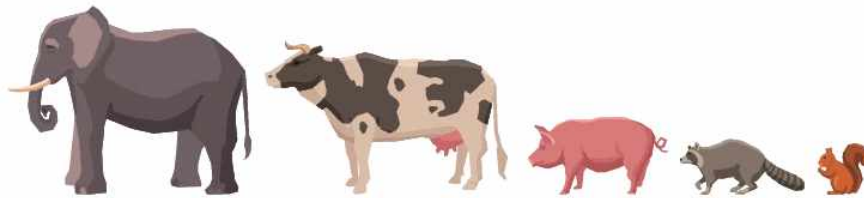
Section 01 정렬의 기본

정렬의 개념

중요 알고리즘 중 하나인 정렬(Sort)은 자료들을 일정한 순서대로 나열한 것



(a) 작은 키에서 큰 키 순으로 오름차순 정렬



(b) 무거운 순에서 가벼운 순으로 내림차순 정렬

그림 11-1 오름차순 정렬과 내림차순 정렬 예

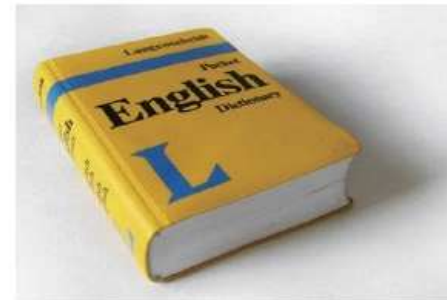


그림 11-2 실생활 정렬 예

정렬의 종류

정렬 알고리즘의 종류

- 선택 정렬(Selection Sort)
- 삽입 정렬(Insertion Sort)
- 버블 정렬(Bubble Sort)
- 퀵 정렬(Quick Sort)



(a) 선택 정렬할 데이터
그림 11-6 선택 정렬 전 초기 상태



(a) 삽입 정렬할 데이터
그림 11-20 삽입 정렬 전 초기 상태



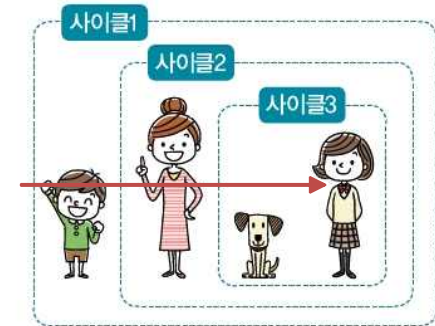
이 그룹을 다시 정렬

그림 12-10 퀵 정렬의 개념

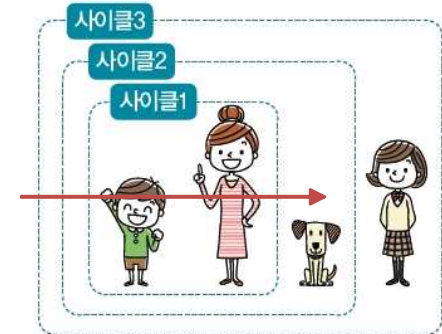
이 그룹을 다시 정렬



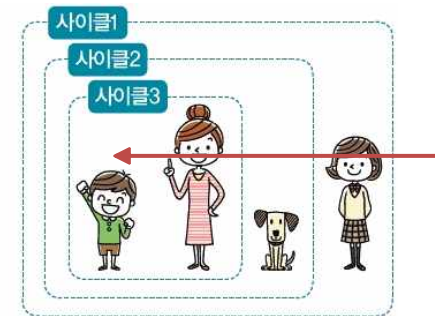
(a) 버블 정렬할 데이터
그림 12-2 버블 정렬 예



(b) 3회 사이클



(b) 3회 사이클



(b) 3회 사이클

Section 01 정렬의 기본

정렬 알고리즘의 종류

- 오름차순 정렬이든 내림차순 정렬이든 결과의 형태만 다를 뿐이지 같은 방식으로 처리됨
- 정렬하는 방법에 대한 정렬 알고리즘은 수십 가지

- 선택 정렬(Selection Sort)
- 삽입 정렬(Insertion Sort)
- 버블 정렬(Bubble Sort)
- 퀵 정렬(Quick Sort)



(a) 선택 정렬할 데이터

그림 11-6 선택 정렬 전 초기 상태



(b) 3회 사이클

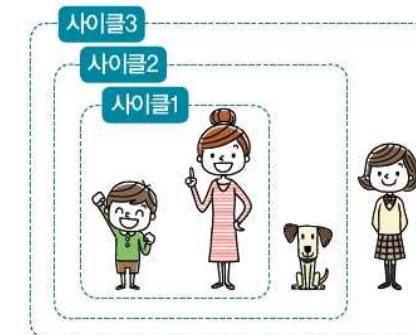
Seoul

English



(a) 삽입 정렬할 데이터

그림 11-20 삽입 정렬 전 초기 상태



(b) 3회 사이클

Section 02 기본 정렬 알고리즘의 원리와 구현

Section 02 기본 정렬 알고리즘의 원리와 구현

1.1 선택 정렬(selection sort) 기초 원리

선택 정렬의 개념 : 여러 데이터 중에서 **가장 작은 값**을 뽑는 작업을 반복하여 값을 정렬
가족을 선택 정렬 방법으로 키 순으로 세우는 과정 예

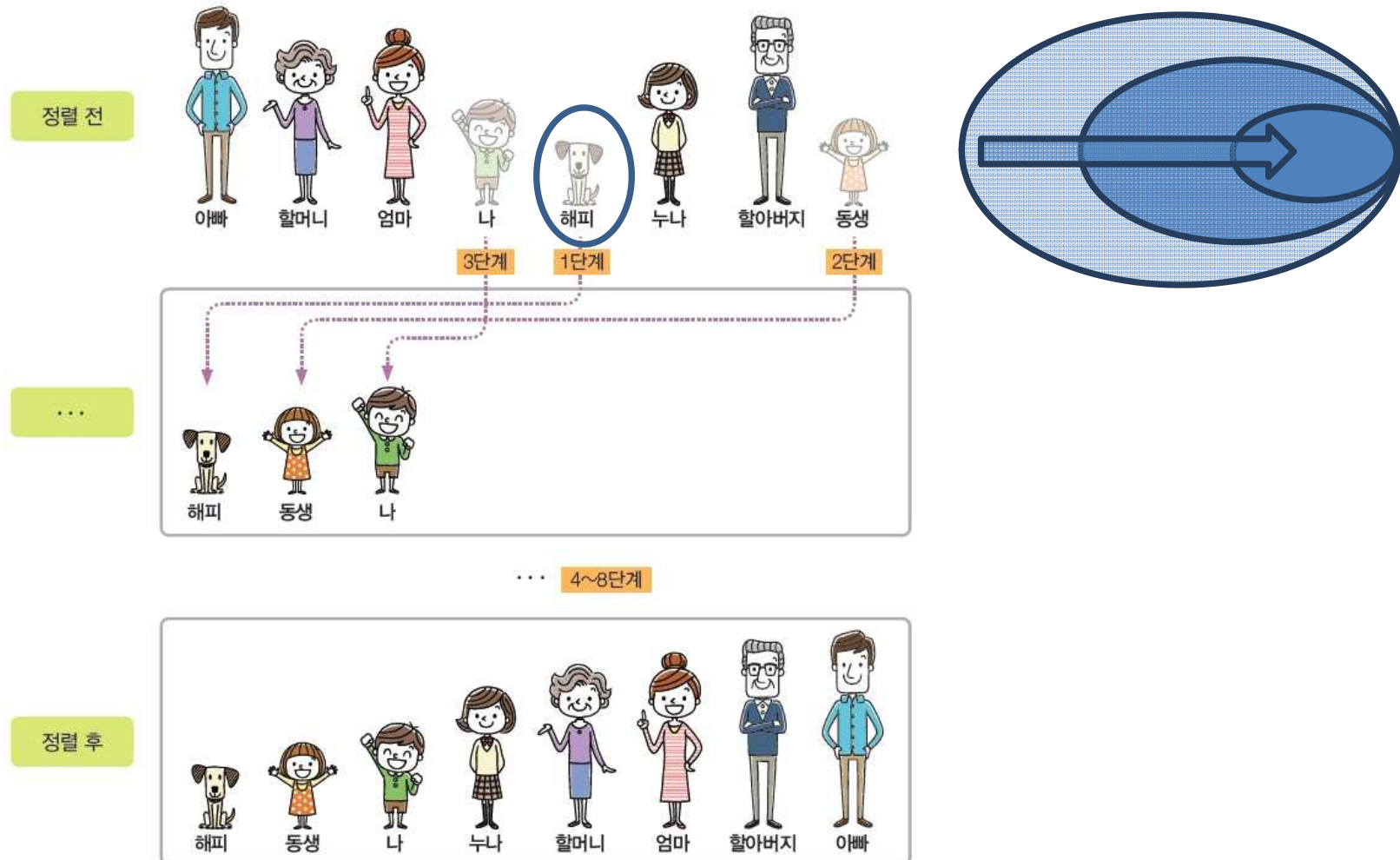


그림 11-3 선택 정렬 예 : 오름차순 정렬

Section 02 기본 정렬 알고리즘의 원리와 구현

최솟값을 찾는 방법

- 1 배열의 첫 번째 값을 가장 작은 값으로 지정한다.



- 2 가장 작은 값으로 지정한 값을 다음 차례의 값과 비교하여 가장 작은 값을 변경하거나 그대로 두는 방법으로
- 3 마지막 값까지 비교를 마친 후 현재 가장 작은 값으로 지정된 값을 가장 작은 값으로 결정한다.

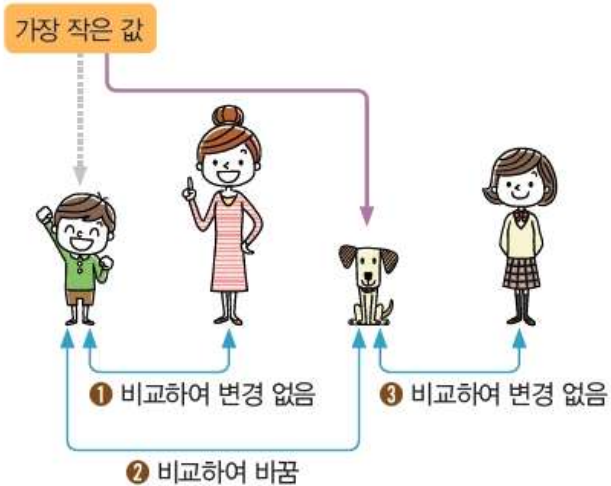


그림 11-4 최솟값을 찾는 단계 예

Section 02 기본 정렬 알고리즘의 원리와 구현

Code11-01.py 배열에서 최솟값 위치를 찾는 함수

```
1 def findMinIdx(ary) :  
2     minIdx = 0 ❶  
3     for i in range(1, len(ary)) :  
4         if (ary[minIdx] > ary[i]) : ❷  
5             minIdx = i  
6     return minIdx  
7  
8 testAry = [55, 88, 33, 77]  
9 minPos = findMinIdx(testAry)  
10 print('최솟값 -->', testAry[minPos])
```

실행 결과

최솟값 --> 33

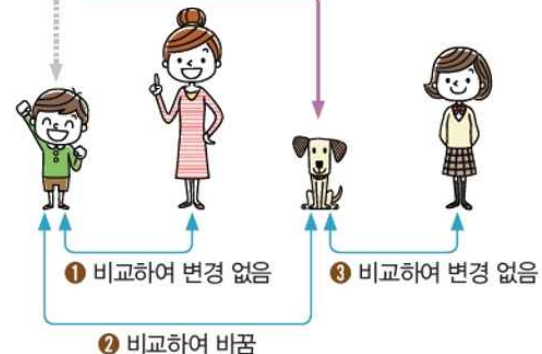
❶ 배열의 첫 번째 값을 가장 작은 값으로 지정한다.

가장 작은 값



❷

가장 작은 값



Section 02 기본 정렬 알고리즘의 원리와 구현

SELF STUDY 11-1

Code11-01.py를 수정해서 최댓값 위치를 찾도록 코드를 작성하자.

실행 결과

최댓값 -> 88

Section 02 기본 정렬 알고리즘의 원리와 구현

선택 정렬 구현

크기가 3인 배열을 준비하고 값을 입력하는 방법을 사용했음

```
ary = [None for _ in range(3)]  
ary[0] = 100  
ary[1] = 200  
ary[2] = 300
```

배열 크기를 미리 지정하지 않고 파이썬에서 제공하는 가변 크기의 배열을 사용

```
ary = []  
ary.append(100)  
ary.append(200)  
ary.append(300)
```

Code11-02.py 선택 정렬의 구현

```
1  ## 클래스와 함수 선언 부분 ##  
2  def findMinIdx(ary):  
3      minIdx = 0  
4      for i in range(1, len(ary)):  
5          if (ary[minIdx] > ary[i]):  
6              minIdx = i  
7      return minIdx  
8
```

Section 02 기본 정렬 알고리즘의 원리와 구현

```
9  ## 전역 변수 선언 부분 ##
10 before = [188, 162, 168, 120, 50, 150, 177, 105]
11 after = []
12
13 ## 메인 코드 부분 ##
14 print('정렬 전 -->', before)
15 for _ in range(len(before)) :
16     minPos = findMinIdx(before)
17     after.append(before[minPos])
18     del(before[minPos])
19 print('정렬 후 -->', after)
```

실행 결과

정렬 전 --> [188, 162, 168, 120, 50, 150, 177, 105]

정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]

Section 02 기본 정렬 알고리즘의 원리와 구현

두 변수 값 교환

알고리즘을 구현할 때는 두 변수 값을 교환해야 하는 경우가 종종 생기는데, 원칙적으로 한 번에 두 변수의 값을 교환할 수 없으므로, 임시 공간을 사용해야 함

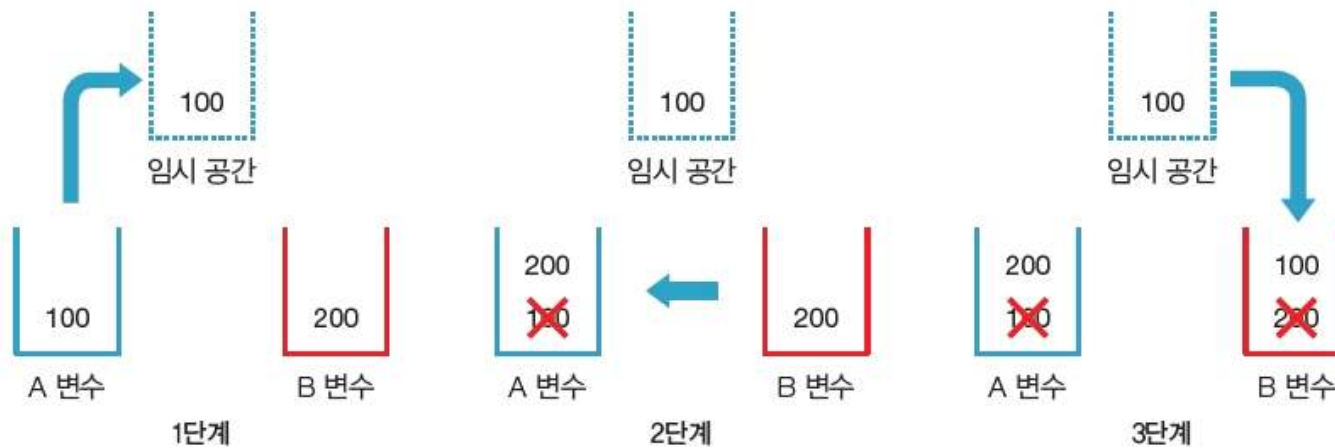


그림 11-5 두 변수 값의 교환

```
temp = A  
A = B  
B = temp
```

또는

```
A, B = B, A
```

Section 02 기본 정렬 알고리즘의 원리와 구현

1.2 선택 정렬 구현 (데이터 4개를 정렬하는 예)

데이터가 4개이므로 (b)와 같이 총 3회의 사이클이 필요함



(a) 선택 정렬할 데이터

그림 11-6 선택 정렬 전 초기 상태



(b) 3회 사이클

Section 02 기본 정렬 알고리즘의 원리와 구현

[사이클1]

먼저 사이클1 중 맨 앞의 값을 가장 작은 값으로 지정한 후 나머지 값과 비교해서 제일 작은 값을 찾음

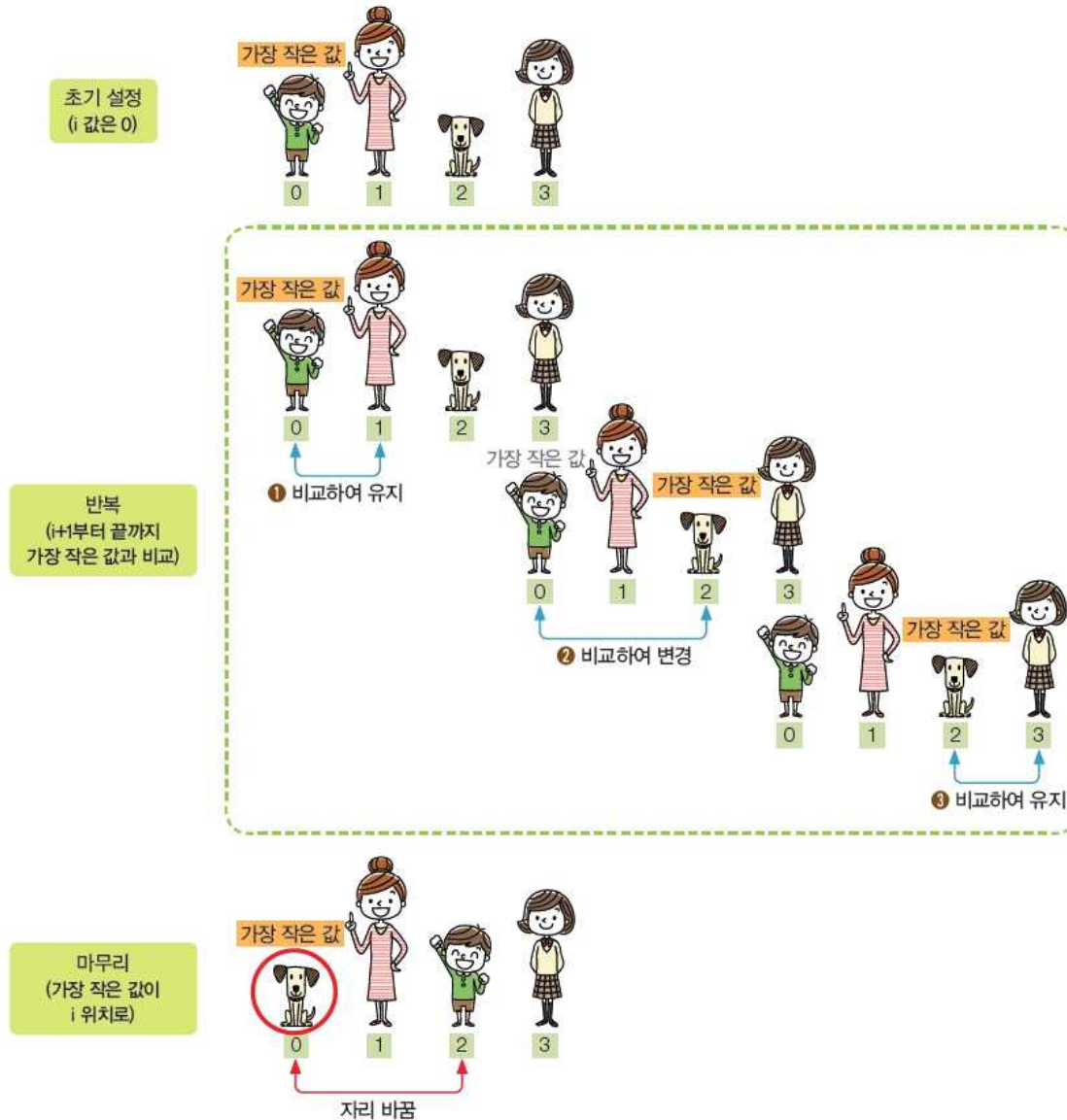


그림 11-7 선택 정렬 예: 사이클1

Section 02 기본 정렬 알고리즘의 원리와 구현

[사이클2]

사이클1에서 찾은 가장 작은 값을 제외한 사이클2 중 맨 앞의 값을 가장 작은 값으로 우선 지정하고, 나머지 값들과 비교하여 제일 작은 값을 찾음

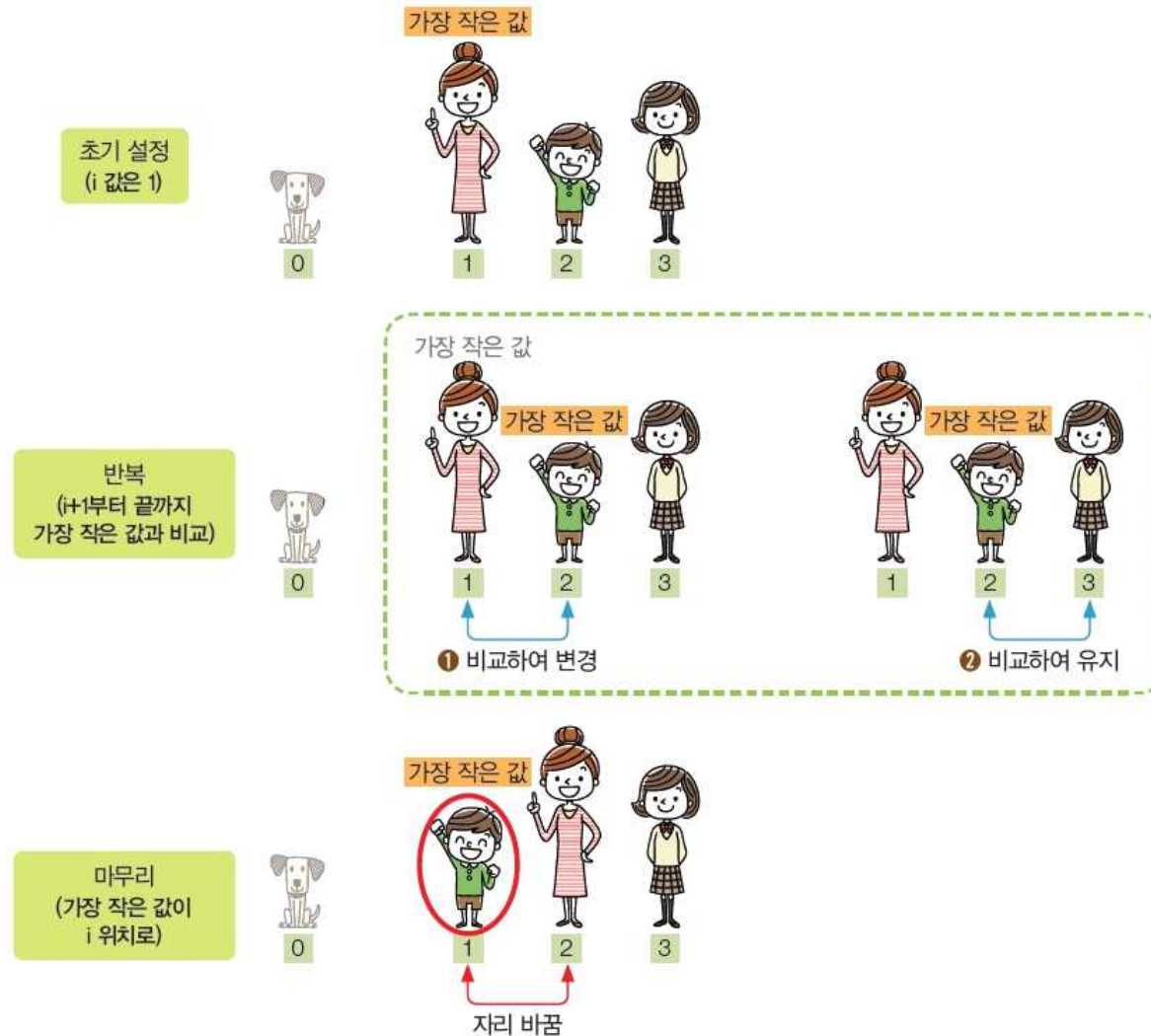


그림 11-8 선택 정렬 예: 사이클2

Section 02 기본 정렬 알고리즘의 원리와 구현

[사이클3]

사이클2에서 찾은 가장 작은 값을 제외한 사이클3 중 맨 앞의 값을 가장 작은 값으로 우선 지정하고, 나머지 값들과 비교하여 맨 앞의 값을 작은 값으로 우선 지정하고, 나머지



그림 11-9 선택 정렬 예 : 사이클3

12장. 선택 정렬 요약

배열에서 최솟값 위치를 찾는 함수

```

1 def findMinIdx(ary):
2     minIdx = 0 ❶
3     for i in range(1, len(ary)):
4         if (ary[minIdx] > ary[i]):
5             minIdx = i
6     return minIdx
7
8 testAry = [55, 88, 33, 77]
9 minPos = findMinIdx(testAry)
10 print('최솟값 -->', testAry[minPos])
    
```



(a) 선택 정렬할 데이터

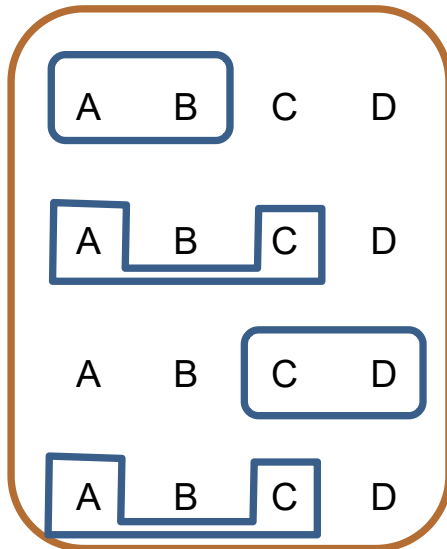
그림 11-6 선택 정렬 전 초기 상태



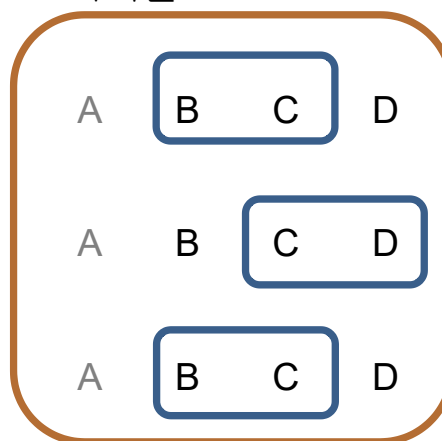
(b) 3회 사이클

A B C D

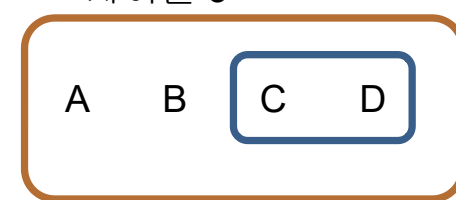
사이클 1



사이클 2



사이클 3



Section 02 기본 정렬 알고리즘의 원리와 구현

모든 사이클 완료 후 정렬된 결과 →



그림 11-10 선택 정렬 후 데이터

Code11-03.py 개선된 선택 정렬

```
1  ## 클래스와 함수 선언 부분 ##
2  def selectionSort(ary) :
3      n = len(ary)
4      for i in range(0, n-1) :
5          minIdx = i
6          for k in range(i+1, n) :
7              if (ary[minIdx] > ary[k]) :
8                  minIdx = k
9          tmp = ary[i]
10         ary[i] = ary[minIdx]
11         ary[minIdx] = tmp
12
13     return ary
14
15 ## 전역 변수 선언 부분 ##
16 dataAry = [188, 162, 168, 120, 50, 150, 177, 105]
17
18 ## 메인 코드 부분 ##
19 print('정렬 전 -->', dataAry)
20 dataAry = selectionSort(dataAry)
21 print('정렬 후 -->', dataAry)
```

실행 결과

정렬 전 --> [188, 162, 168, 120, 50, 150, 177, 105]
정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]

Section 02 기본 정렬 알고리즘의 원리와 구현

선택 정렬 성능

- 정렬에서 중요한 사항 중 하나는 정렬을 완료하는 비교 횟수임
- Code11-03.py의 7행이 몇 번 수행되었는지 확인하는 예

i 값	k 값	비교 횟수
0	1, 2, 3	3회
1	2, 3	2회
2	3	1회

i가 0일 때 → $3(=4-1)$ 번 수행
i가 1일 때 → $2(=4-2)$ 번 수행
i가 2일 때 → $1(=4-3)$ 번 수행

데이터 개수가 4일 때 7행은 이와 같이 반복

i가 0일 때 → $n-1$ 번 수행
i가 1일 때 → $n-2$ 번 수행
i가 2일 때 → $n-3$ 번 수행
i가 3일 때 → $n-4$ 번 수행
...(중략)...
i가 $n-3$ 일 때 → 2번 수행
i가 $n-2$ 일 때 → 1번 수행

데이터 개수가 n 개라면 이와 같이 계산됨

Section 02 기본 정렬 알고리즘의 원리와 구현

- 비교 횟수는 거꾸로 하면 $1+2+3+\dots+(n-1)$ 번이 되는데 이를 수식으로 유도하기 전에 1부터 10까지 합계를 구하는 방법

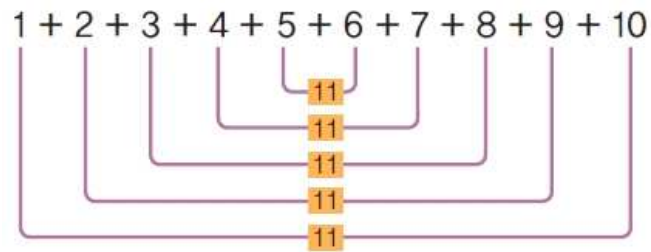


그림 11-11 1부터 10까지 합계

$$11 * 5\text{회} = 55$$

11이 5회 반복되므로 이와 같이 계산됨

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

숫자 10을 중심으로 표현

- 결국 1부터 n 까지 합계는 다음 수식과 같음(10 대신에 n 을 대입)

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

- 비교 횟수의 합계인 $1+2+3+\dots+(n-1)$ 은 1부터 $n-1$ 까지 합계이므로 위 수식에서 n 대신에 $n-1$ 을 대입

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

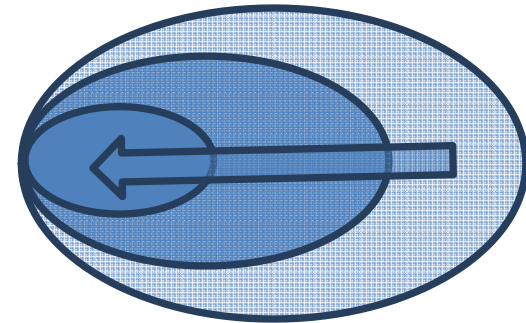
Section 02 기본 정렬 알고리즘의 원리와 구현

2.1 삽입 정렬(insertion sort) 기초 원리

삽입 정렬의 개념 : 기존 데이터 중에서 자신의 위치를 찾아 데이터를 삽입하는 정렬
가족을 키 순으로 세우는 예에 삽입 정렬 적용



그림 11-12 삽입 정렬 전 초기 상태



가장 앞에 있는 아빠를 일단 줄에 세움

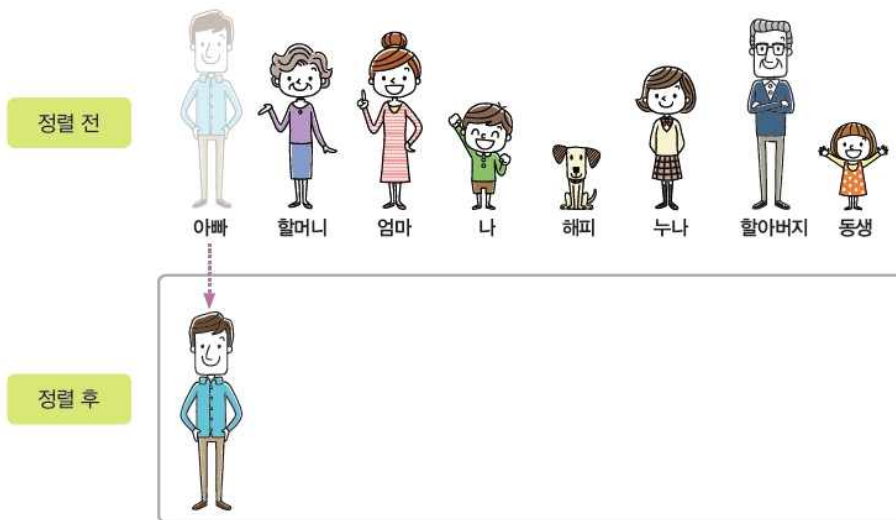


그림 11-13 삽입 정렬 1단계: 아빠 정렬

Section 02 기본 정렬 알고리즘의 원리와 구현

할머니는 아빠보다 작으므로 아빠 앞에 세움

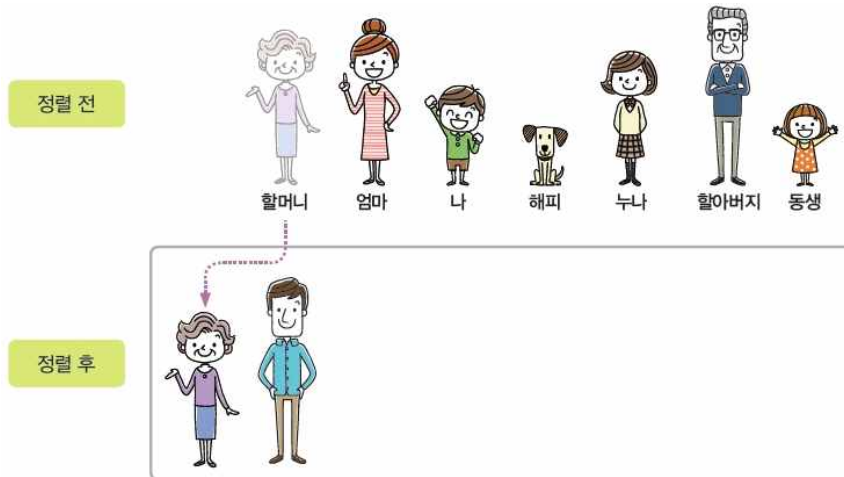


그림 11-14 삽입 정렬 2단계: 할머니 정렬

다음으로 엄마를 줄에 세움. 엄마는 할머니보다 크고, 아빠보다 작으므로 그 사이에 세움

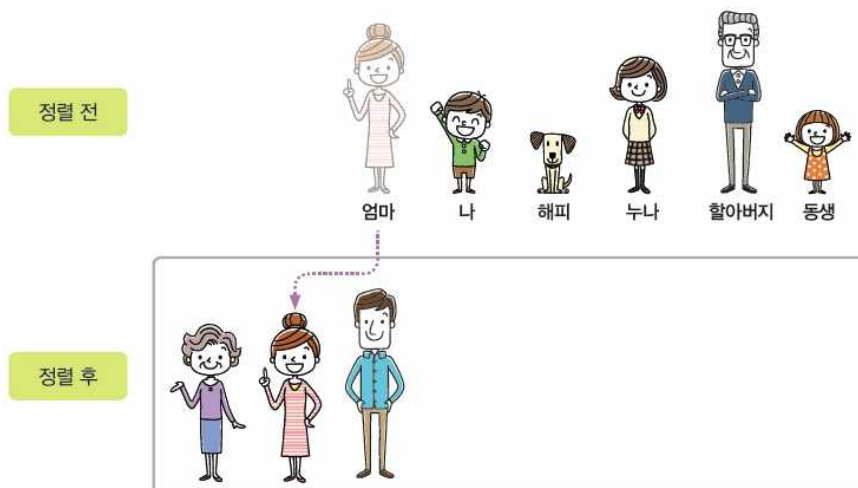


그림 11-15 삽입 정렬 3단계: 엄마 정렬

Section 02 기본 정렬 알고리즘의 원리와 구현

같은 방식으로 각 가족을 자신보다 작은 사람과 큰 사람 사이에 세우면 됨 → 가족을 키 순서대로 오름차순 정렬

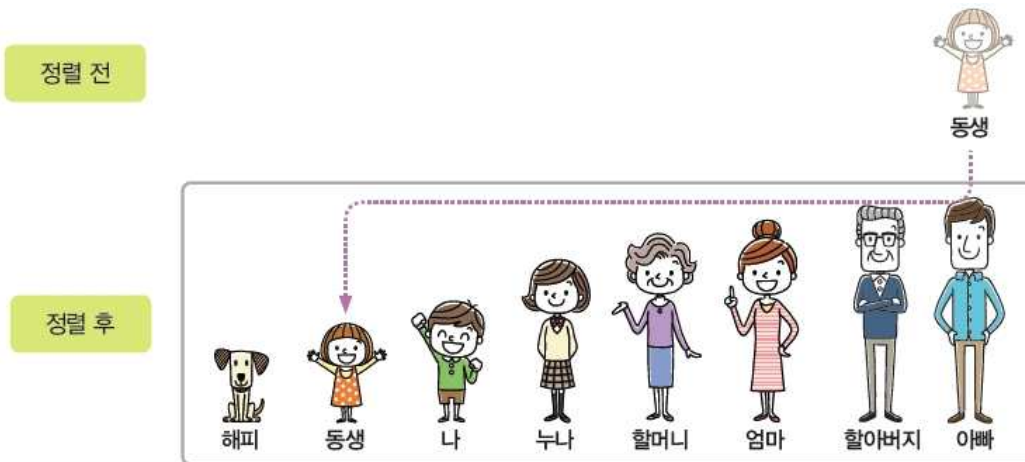


그림 11-16 삽입 정렬 4~8단계 : 마지막 동생 정렬

Section 02 기본 정렬 알고리즘의 원리와 구현

삽입 위치를 찾는 방법

빈 배열일 때는 첫 번째 자리에 삽입함

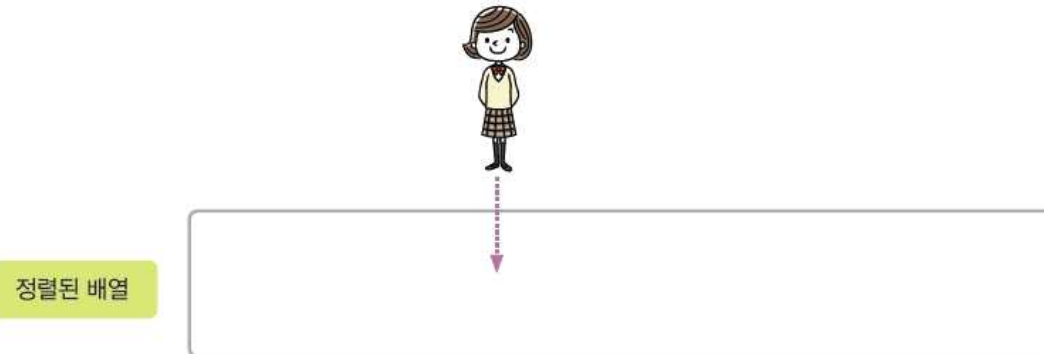


그림 11-17 빈 배열일 때

배열에 삽입할 값보다 큰 값이 있을 때는 처음부터 비교해 가면서 자신보다 큰 값을 만나면 그 값 바로 앞에 삽입

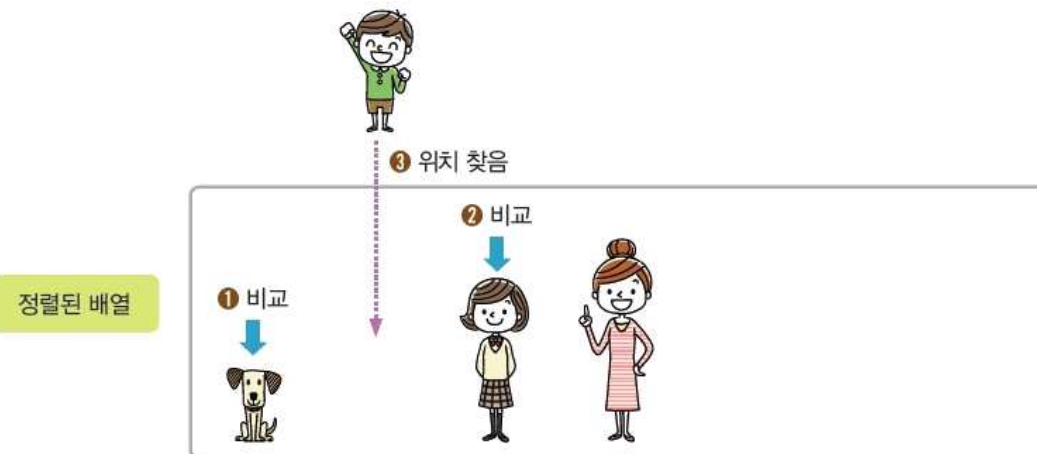


그림 11-18 배열에 삽입할 값보다 큰 값이 있을 때

Section 02 기본 정렬 알고리즘의 원리와 구현

배열에 삽입할 값보다 큰 값이 없을 때는 맨 뒤에 삽입

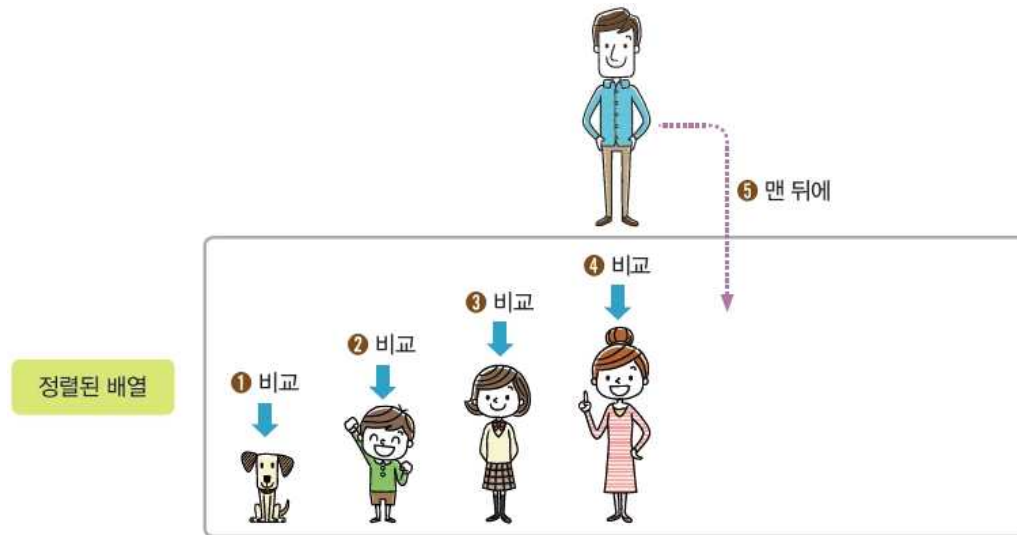


그림 11-19 배열에 삽입할 값보다 큰 값이 없을 때

Section 02 기본 정렬 알고리즘의 원리와 구현

Code11-04.py 배열에서 자신이 삽입될 위치를 찾는 함수

```
1 def findInsertIdx(ary, data) :
2     findIdx = -1          # 초깃값은 없는 위치로
3     for i in range(0, len(ary)) :
4         if (ary[i] > data) :
5             findIdx = i
6             break
7     if findIdx == -1 :      # 큰 값을 못 찾음 == 제일 마지막 위치
8         return len(ary)
9     else :
10        return findIdx
11
12 testAry = []
13 insPos = findInsertIdx(testAry, 55)
14 print('삽입할 위치 -->', insPos)
15
16 testAry = [33, 77, 88]
17 insPos = findInsertIdx(testAry, 55)
18 print('삽입할 위치 -->', insPos)
19
20 testAry = [33, 55, 77, 88]
21 insPos = findInsertIdx(testAry, 100)
22 print('삽입할 위치 -->', insPos)
```

실행 결과

```
삽입할 위치 --> 0
삽입할 위치 --> 1
삽입할 위치 --> 4
```

Section 02 기본 정렬 알고리즘의 원리와 구현

삽입 정렬 구현

파이썬에서 제공하는 insert(삽입할 위치, 값) 함수를 사용하면 간단

```
testAry = []  
testAry.insert(0, 55)  
testAry
```

빈 배열에는 x번째 위치에 값을 삽입

실행 결과

```
[55]
```

```
testAry = [33, 77, 88]  
testAry.insert(1, 55)  
testAry
```

1번째 위치에 55를 삽입

실행 결과

```
[33, 55, 77, 88]
```

```
testAry = [33, 55, 77, 88]  
testAry.insert(4, 100)  
testAry
```

배열의 맨 뒤에 값을 삽입
4번째 위치에 100을 삽입

실행 결과

```
[33, 55, 77, 88, 100]
```


Section 02 기본 정렬 알고리즘의 원리와 구현

Code11-05.py 삽입 정렬의 구현

```
1  ## 클래스와 함수 선언 부분 ##
2  def findInsertIdx(ary, data) :
3      findIdx = -1          # 초깃값은 없는 위치로
4      for i in range(0, len(ary)) :
5          if (ary[i] > data) :
6              findIdx = i
7              break
8      if findIdx == -1 :      # 큰 값을 못 찾음 == 제일 마지막 위치
9          return len(ary)
10     else :
11         return findIdx
12
13 ## 전역 변수 선언 부분 ##
14 before = [188, 162, 168, 120, 50, 150, 177, 105]
15 after = []
16
17 ## 메인 코드 부분 ##
18 print('정렬 전 -->', before)
19 for i in range(len(before)) :
20     data = before[i]
21     insPos = findInsertIdx(after, data)
22     after.insert(insPos, data)
23 print('정렬 후 -->', after)
```

실행 결과

정렬 전 --> [188, 162, 168, 120, 50, 150, 177, 105]

정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]

Section 02 기본 정렬 알고리즘의 원리와 구현

SELF STUDY 11-2

Code11-05.py를 수정해서 랜덤하게 0~200 사이의 숫자 10개를 생성한 후, 내림차순으로 정렬하도록 코드를 작성
하자(실행 결과는 실행할 때마다 다름).

실행 결과

정렬 전 --> [107, 152, 136, 128, 176, 24, 97, 13, 102, 137]

정렬 후 --> [176, 152, 137, 136, 128, 107, 102, 97, 24, 13]

Section 02 기본 정렬 알고리즘의 원리와 구현

2.2 삽입 정렬 구현

- 배열을 2개 사용하는 것보다 배열 하나에서 데이터를 정렬하는 방식이 더 효율적임
- 간단히 데이터 4개를 정렬한 예(총 3회의 사이클이 필요)

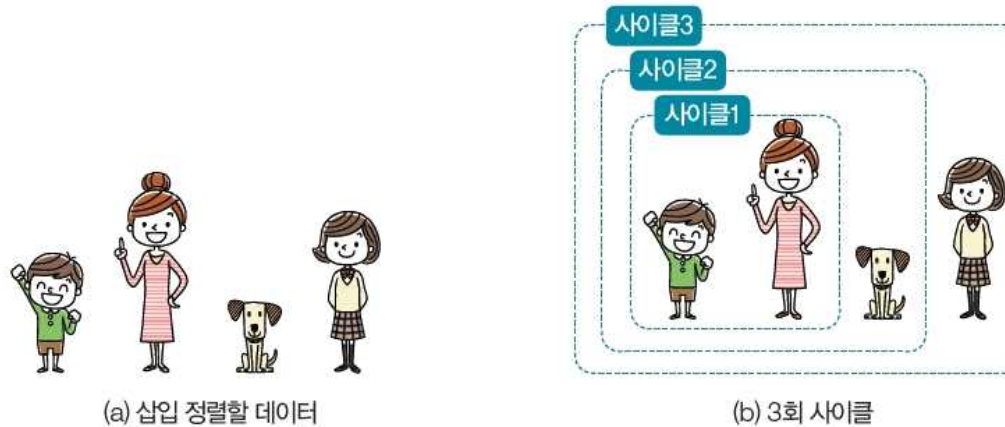


그림 11-20 삽입 정렬 전 초기 상태

- 먼저 [사이클1]의 마지막을 현재로 두고 두 데이터를 비교해서 작은 것을 앞으로 가져옴

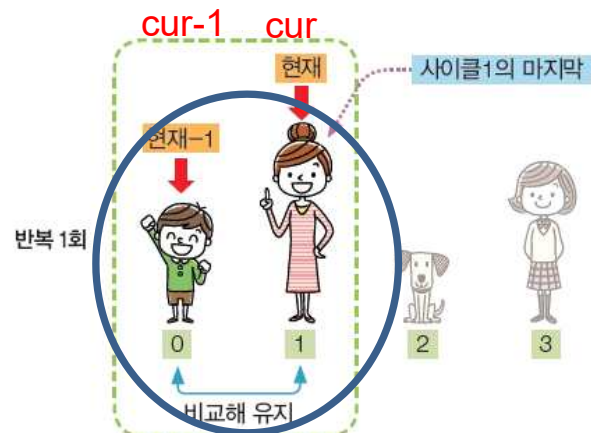


그림 11-21 삽입 정렬 예: 사이클1

Section 02 기본 정렬 알고리즘의 원리와 구현

[사이클2] 사이클중 마지막 2개부터 각 쌍을 비교해서 작은 것을 앞으로 가져옴

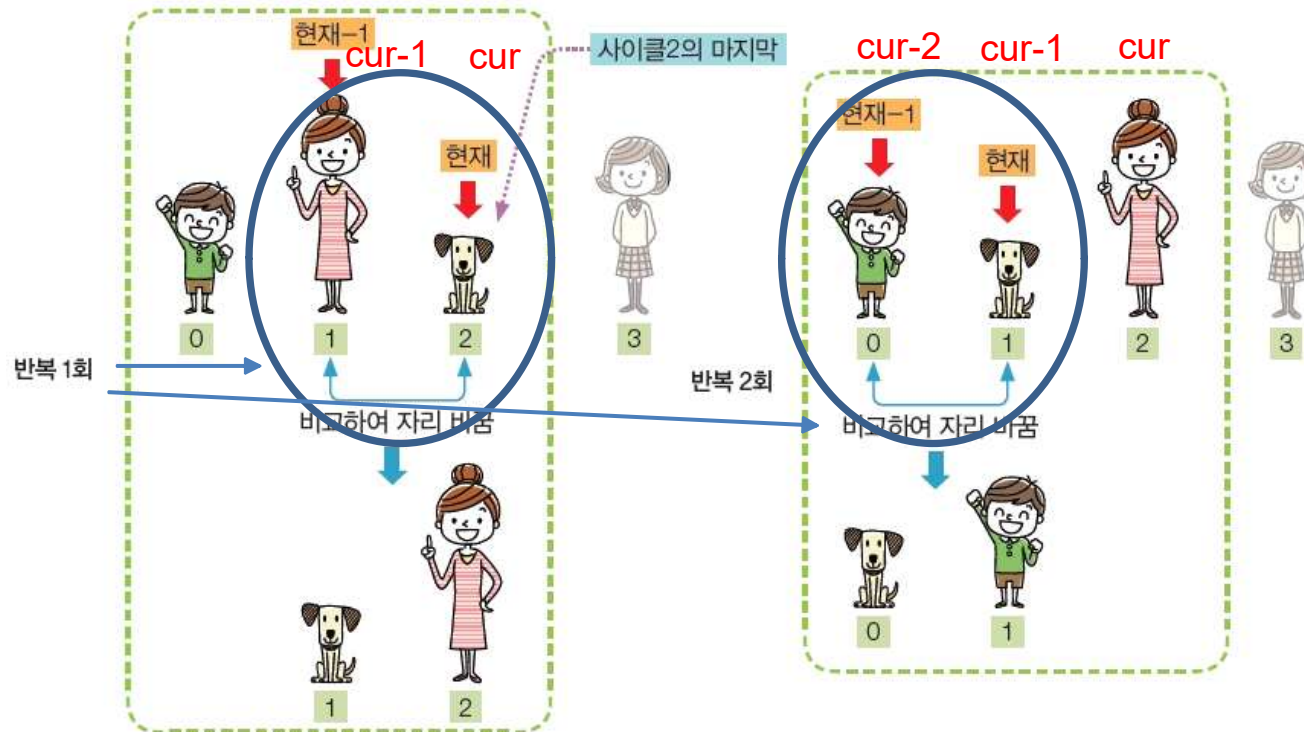
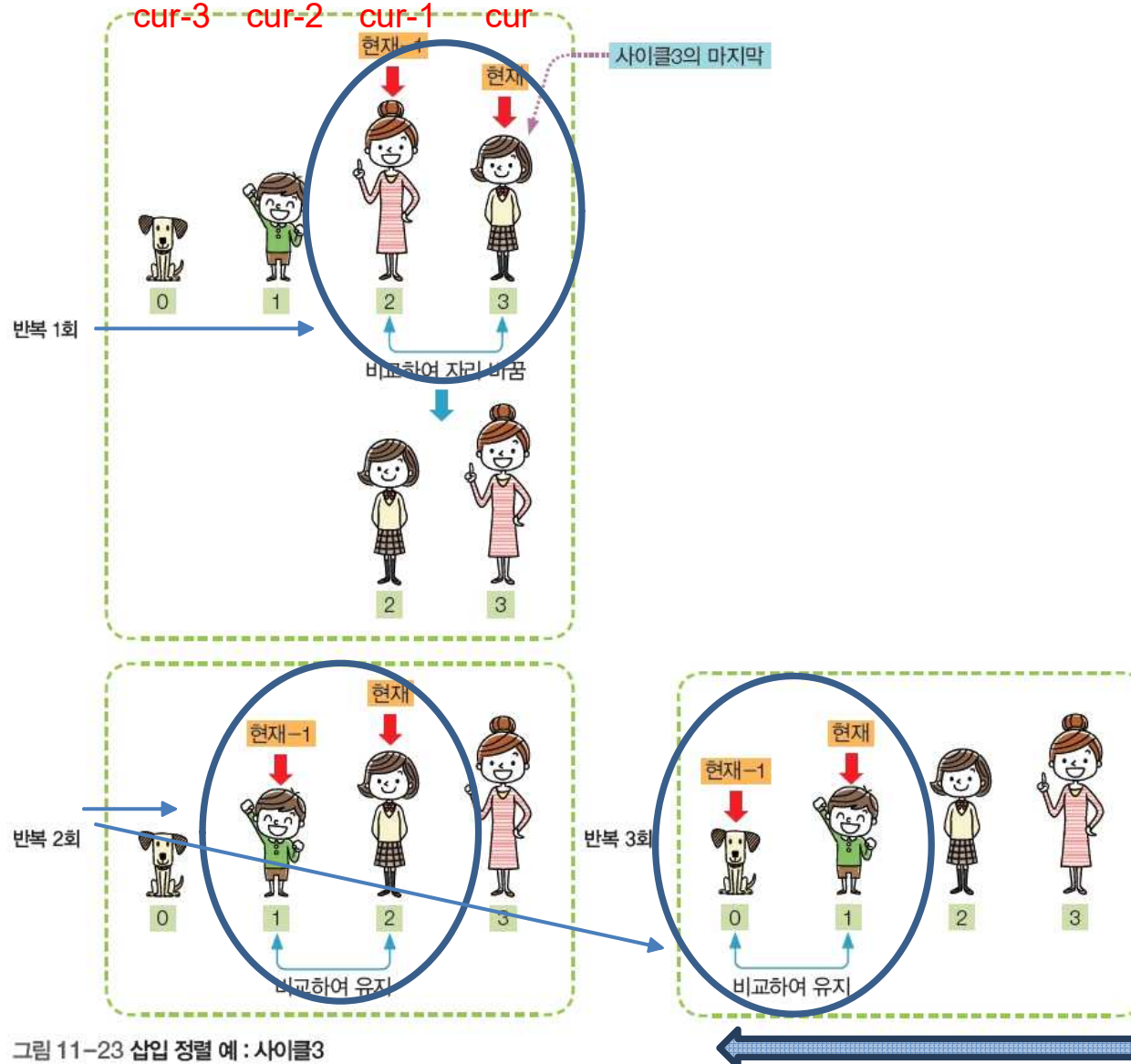


그림 11-22 삽입 정렬 예 : 사이클2

Section 02 기본 정렬 알고리즘의 원리와 구현

[사이클3] 사이클3 중 마지막 2개부터 각 쌍을 비교해서 작은 것을 앞으로 가져옴



삽입 정렬 요약



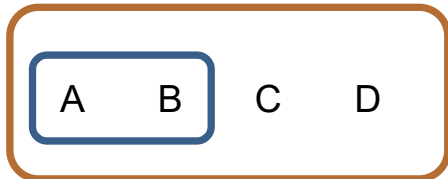
(a) 삽입 정렬할 데이터



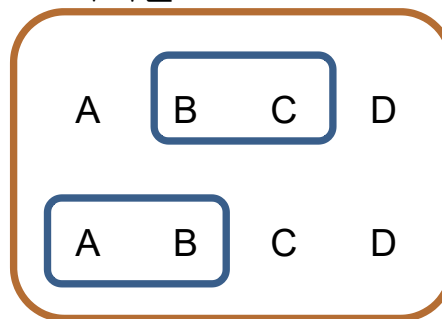
(b) 3회 사이클

A B C D

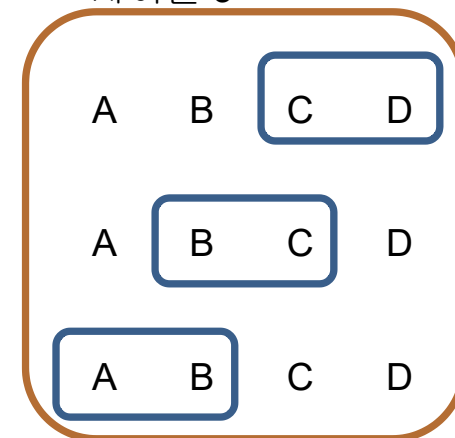
사이클 1



사이클 2



사이클 3



Section 02 기본 정렬 알고리즘의 원리와 구현

모든 사이클이 완료된 후 정렬 결과



그림 11-24 정렬 후 데이터

Code11-06.py 삽입 정렬의 효율적인 구현

```
1  ## 클래스와 함수 선언 부분 ##
2  def insertionSort(ary) :
3      n = len(ary)
4      for end in range(1, n) :
5          for cur in range(end, 0, -1) :
6              if( ary[cur-1] > ary[cur]) :
7                  ary[cur-1], ary[cur] = ary[cur], ary[cur-1]
8      return ary
9
10 ## 전역 변수 선언 부분 ##
11 dataAry = [188, 162, 168, 120, 50, 150, 177, 105]
12
13 ## 메인 코드 부분 ##
14 print('정렬 전 -->', dataAry)
15 dataAry = insertionSort(dataAry)
16 print('정렬 후 -->', dataAry)
```

실행 결과

정렬 전 --> [188, 162, 168, 120, 50, 150, 177, 105]

정렬 후 --> [50, 105, 120, 150, 162, 168, 177, 188]

Section 02 기본 정렬 알고리즘의 원리와 구현

- 삽입 정렬 성능

- 삽입 정렬도 선택 정렬과 마찬가지로 연산 수는 $O(n^2)$
- 입력 개수가 커질수록 기하급수적으로 비교 횟수(또는 연산 횟수)가 늘어나기에 성능이 좋지 않은 알고리즘

Section 03 기본 정렬 알고리즘의 응용

Section 03 기본 정렬 알고리즘의 응용

1차원 배열의 중앙값 계산

평균값과 중앙값

평균값은 전체 데이터 값을 합친 후 개수로 나누는 것

데이터 값 중에서 비정상적인 수치가 섞여 있는 예(한 학생만 1원 단위로 적고 나머지는 1만 원 단위로 적은 경우)

7	5	11	6	9	80000	10	6	15	12
---	---	----	---	---	-------	----	---	----	----

중앙값 계산

중앙값은 데이터를 일렬로 정렬해서 나열한 후 나열된 숫자의 가운데에 위치하는 값을 대푯값으로 하는 방법

용돈의 중앙값을 처리하기 위해 정렬한 경우

5	6	6	7	9	10	11	12	15	80000
---	---	---	---	---	----	----	----	----	-------

Section 03 기본 정렬 알고리즘의 응용

Code11-07.py 중앙값 계산

```
1  ## 클래스와 함수 선언 부분 ##
2  def selectionSort(ary) :
3      n = len(ary)
4      for i in range(0, n-1) :
5          minIdx = i
6          for k in range(i+1, n) :
7              if (ary[minIdx] > ary[k]) :
8                  minIdx = k
9          tmp = ary[i]
10         ary[i] = ary[minIdx]
11         ary[minIdx] = tmp
12
13     return ary
14
15 ## 전역 변수 선언 부분 ##
16 moneyAry = [7, 5, 11, 6, 9, 80000, 10, 6, 15, 12]
17
18 ## 메인 코드 부분 ##
19 print('용돈 정렬 전 -->', moneyAry)
20 moneyAry = selectionSort(moneyAry)
21 print('용돈 정렬 후 -->', moneyAry)
22 print('용돈 중앙값 --> ', moneyAry[len(moneyAry)//2])
```

실행 결과

용돈 정렬 전 --> [7, 5, 11, 6, 9, 80000, 10, 6, 15, 12]

용돈 정렬 후 --> [5, 6, 6, 7, 9, 10, 11, 12, 15, 80000]

용돈 중앙값 --> 10

Section 03 기본 정렬 알고리즘의 응용

파일 이름의 정렬 출력

지정된 폴더에서 하위 폴더를 포함한 파일 목록 추출

C:\Windows\System32 폴더의 파일을 배열에 저장하는 기능을 하는 코드

```
import os

fnameAry = []
folderName = 'C:/Windows/System32'
for dirName, subDirList, fnames in os.walk(folderName):
    for fname in fnames:
        fnameAry.append(fname)

print(len(fnameAry))
```

실행 결과

9593

파일 목록

```
['accessibilitycpl.dll', '12520437.cpx', '12520850.cpx', 'aadtb.dll', 'BitLockerCsp.dll', 'AarSvc.dll', 'AboveLockAppHost.dll', ...]
```

Section 03 기본 정렬 알고리즘의 응용

파일 목록 정렬

파일은 '파일명.확장명'으로 구분

파일명 및 확장명으로 분리해서 정렬하는 경우

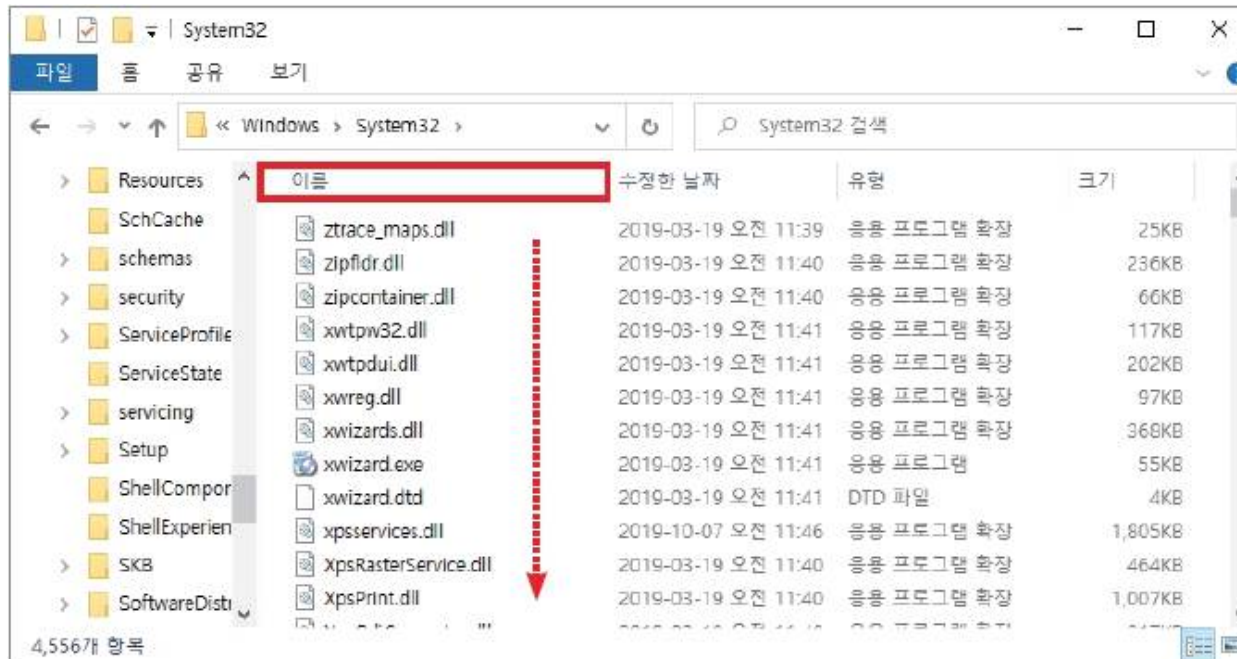


그림 11-25 파일 탐색기에서 파일명을 내림차순으로 정렬한 상태

Section 03 기본 정렬 알고리즘의 응용

Code11-08.py 파일명으로 내림차순 정렬

```
1  import os
2
3  ## 클래스와 함수 선언 부분 ##
4  def makeFileList(folderName) :
5      fnameAry = []
6      for dirName, subDirList, fnames in os.walk(folderName) :
7          for fname in fnames :
8              fnameAry.append(fname)
9      return fnameAry
10
11 def insertionSort(ary) :
12     n = len(ary)
13     for end in range(1, n) :
14         for cur in range(end, 0, -1) :
15             if (ary[cur-1] < ary[cur]) :
16                 ary[cur-1], ary[cur] = ary[cur], ary[cur-1]
17     return ary
18
19 ## 전역 변수 선언 부분 ##
20 fileAry = []
21
22 ## 메인 코드 부분 ##
23 fileAry = makeFileList('C:/Program Files/Common Files')
24 fileAry = insertionSort(fileAry)
25 print('파일명 역순 정렬 -->', fileAry)
```

Section 03 기본 정렬 알고리즘의 응용

실행 결과

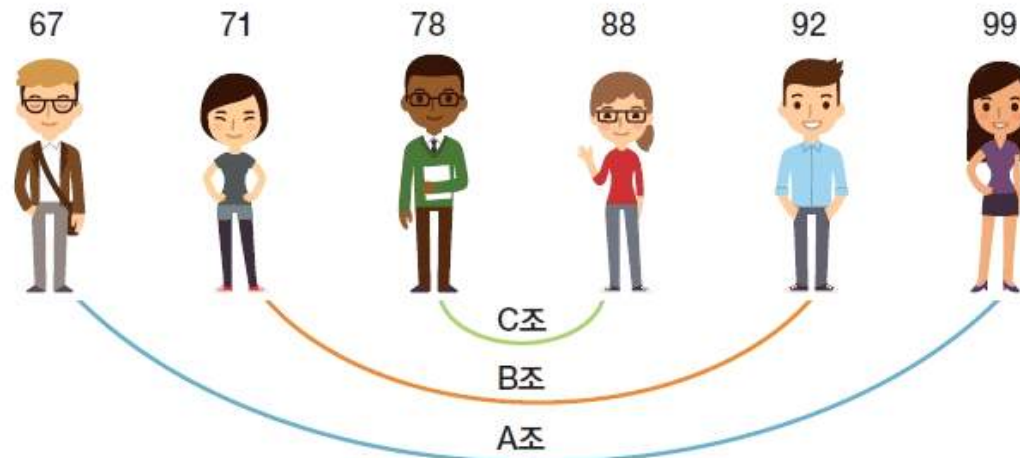
파일명 역순 정렬 --> ['zh-phonetic.xml', 'zh-dayi.xml', 'zh-changjei.xml', 'wab32res.dll.mui', 'wab32res.dll', 'wab32.dll', 'vstoe90.tlb', 'vstoe100.tlb', 'vstoe.dll', 'vsockver.dll', 'vsocklib_x86.dll', 'vsocklib_x64.dll', 'vsock.sys', 'vsock.inf', 'vsock.cat', 'vsjitdebuggerps.dll', 'vmx86ver.dll', 'vmx86.sys', 'vmx86.inf', 'vmx86.cat', 'vmusbver.dll', 'vmusb.sys', (생략)]

응용예제 01 성적별로 조 편성하기

난이도 ★☆☆☆☆

예제 설명

학생의 성적별로 정렬한 후 가장 성적이 높은 학생과 가장 성적이 낮은 학생을 짝으로 조를 만들어 주자. 전체 학생 수는 짝수라고 가정한다.



실행 결과

```
Python
File Edit Shell Debug Options Window Help
RESTART: C:\CookData\Ex11-01.py
성적 정렬 후 -> [['선미', 88], ['초아', 99], ['화사', 71], ['영탁', 78], ['영웅', 67], ['민호', 92]]
정렬 후 -> [['영웅', 67], ['화사', 71], ['영탁', 78], ['선미', 88], ['민호', 92], ['초아', 99]]
## 성적별 조 명칭표 ##
영웅 : 초아
화사 : 민호
영탁 : 선미
>>>
```

응용예제 02 2차원 배열의 중앙값 찾기

난이도 ★☆☆☆☆

예제 설명 2차원 배열 값에서 중앙값을 찾는다. 2차원 배열을 1차원 배열로 만든 후 정렬하는 방법을 사용한다.

55	33	250	44
88	1	76	23
199	222	38	47
155	145	20	99



55	33	250	44	88	1	76	23	199	222	38	47	155	145	20	99
----	----	-----	----	----	---	----	----	-----	-----	----	----	-----	-----	----	----



1	20	23	33	38	44	47	55	67	88	99	145	155	199	222	250
---	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----

실행 결과

```
Python
File Edit Shell Debug Options Window Help
===== RESTART: C:\WookData\WEX11-02.py =====
1차원 변경 후, 정렬 전 --> [55, 33, 250, 44, 88, 1, 67, 23, 199, 222, 38, 47, 155, 145, 20, 99]
1차원 변경 후, 정렬 후 --> [1, 20, 23, 33, 38, 44, 47, 55, 67, 88, 99, 145, 155, 199, 222, 250]
중앙값 --> 67
>>>
```



Thank You
