한국폴리텍대학
로봇캠퍼스

정렬 기본

자료구조와 알고리즘
12주차 강의

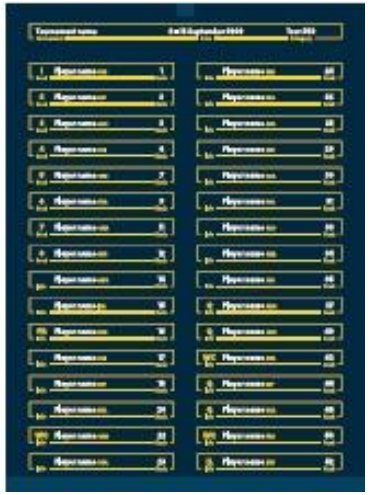
강의 계획표 / C# 프로그래밍 2판

주	주제
1	자료구조와 알고리즘 소개
2	파이썬 기초 문법과 데이터 형식
3	선형 리스트
4	단순 연결 리스트
5	원형 연결 리스트
6	스택
7	큐
8	중간고사
9	이진 트리
10	그래프
11	재귀 호출
12	**정렬 기본**
13	정렬 고급
14	검색
15	동적 계획법
16	기말고사

정렬 기본

정렬이란?

-> 학교 출석부 또는 종류에 따라 가지런히 놓여 있는 칼들처럼 순서대로 데이터가 나열되어 있는 것
-> 정렬을 통해 빠르고 편리하게 사용할 수 있음
-> 자격증 시험 또는 면접 질문 단골

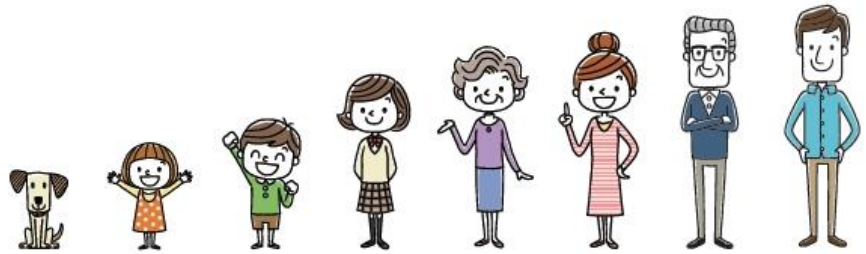


↑ 원하는 자리에 자유롭게 앉을 수 있는 대학교도 출석부에는 학생의 학번 순서 또는 이름 순서로 학생 명단이 작성되어 있음

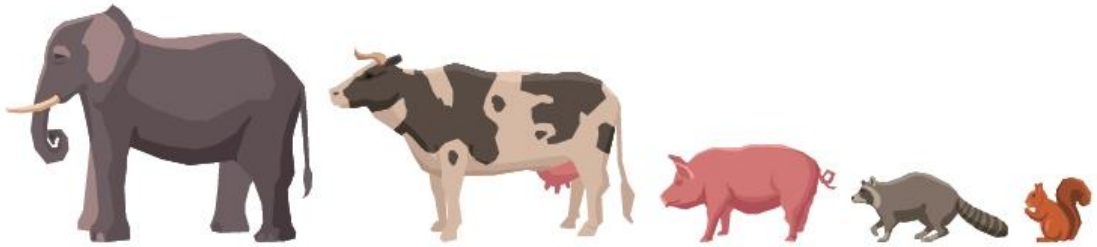
↑ 가지런히 놓고 사용하면 더 편리한 칼

정렬의 개념

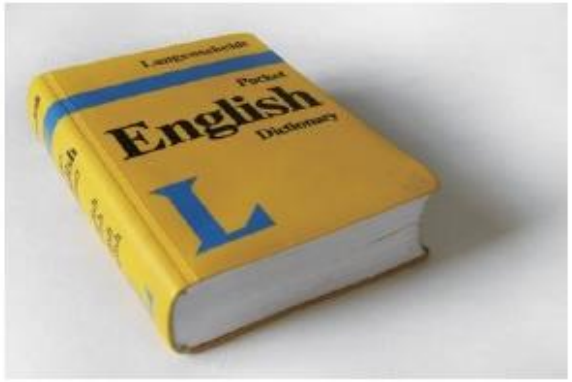
-> 중요 알고리즘 중 하나인 정렬(Soft)은 자료들을 일정한 순서대로 나열한 것



(a) 작은 키에서 큰 키 순으로 오름차순 정렬



(b) 무거운 순에서 가벼운 순으로 내림차순 정렬



정렬 알고리즘의 종류

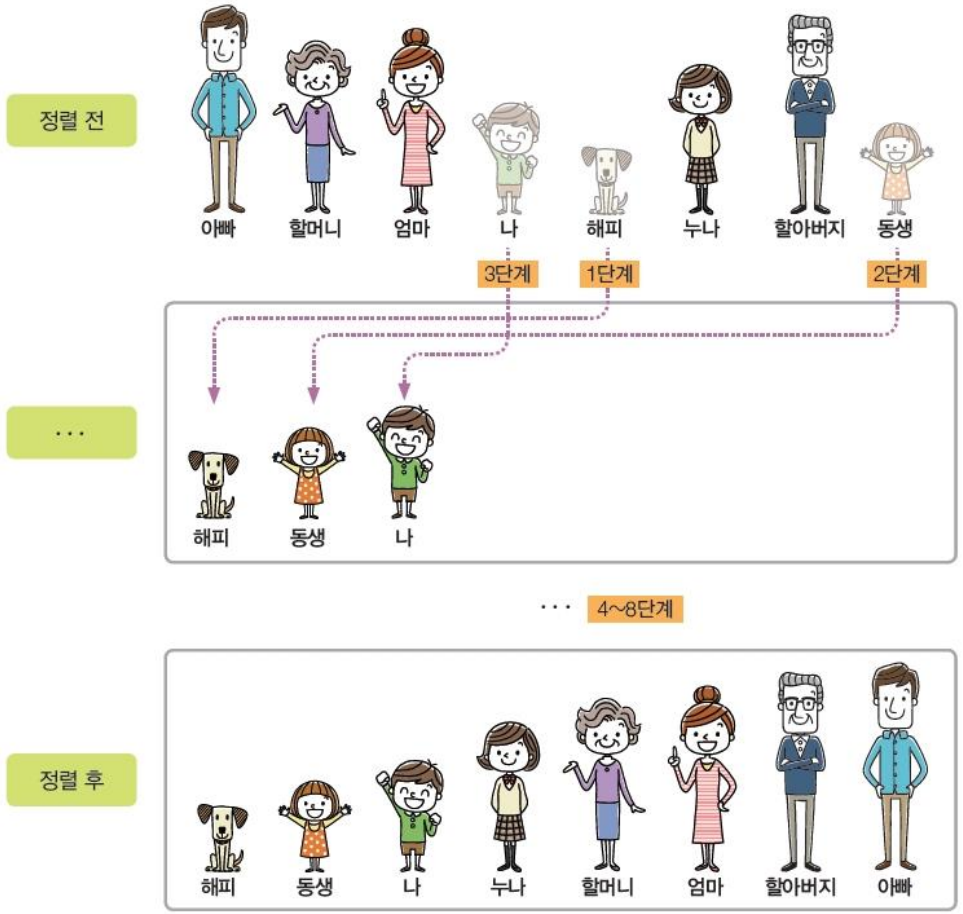
-> 오름차순 정렬이든 내림차순 정렬이든 결과의 형태만 다를 뿐이지 같은 방식으로 처리됨
-> 정렬하는 방법에 대한 정렬 알고리즘은 수십 가지

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

정렬 기본 원리와 구현

선택 정렬

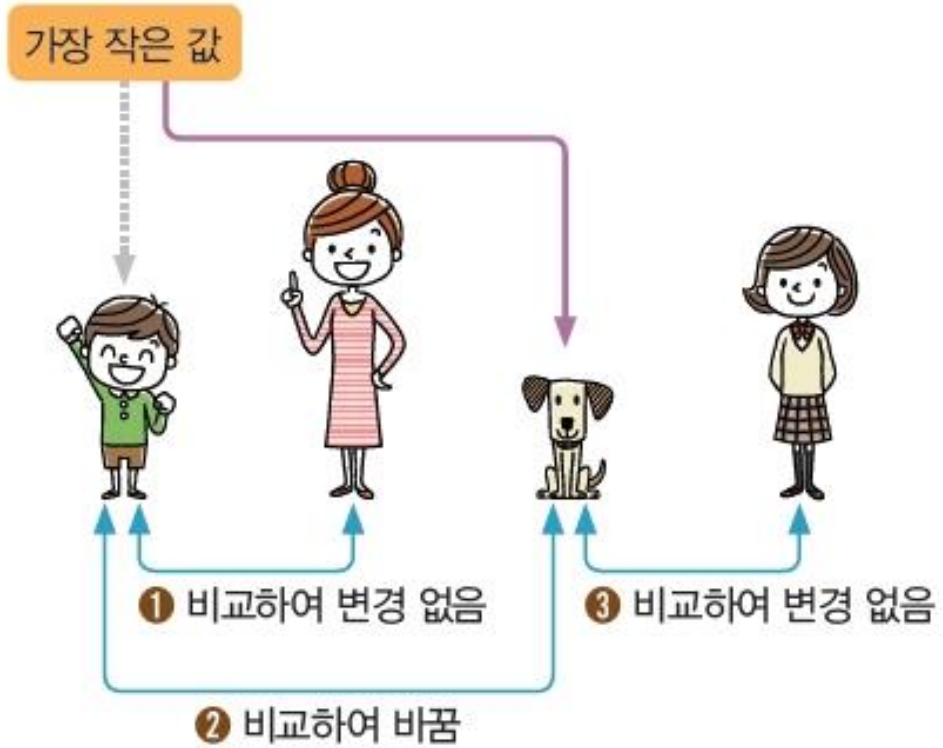
-> 선택 정렬의 개념 : 여러 데이터 중에서 가장 작은 값을 뽑는 작동을 반복하여 값을 정렬



가족을 선택 정렬 방법(오름 차순)으로
키 순으로 세우는 과정 예

최소값을 찾는 방법

❶ 배열의 첫 번째 값을 가장 작은 값으로 지정한다.
❷ 가장 작은 값으로 지정한 값을 다음 차례의 값과 비교하여 가장 작은 값을 변경하거나 그대로 두는 방법으로
❸ 마지막 값까지 비교를 마친 후 현재 가장 작은 값으로 지정된 값을 가장 작은 값으로 결정한다.



최소값을 찾는 함수 코드

```
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])
```

❷

실행 결과

```
최솟값 --> 33
```

Code [illegible] 를 수정해서 최댓값 위치를 찾도록 코드를 작성하자.

실행 결과

```
최댓값 --> 88
```

선택 정렬 구현 코드

```
## 함수 선언 부분 ##
def findMinIdx(ary) :
    minIdx = 0
    for i in range(1, len(ary)) :
        if (ary[minIdx] > ary[i]) :
            minIdx = i
    return minIdx

## 전역 변수 선언 부분 ##
before = [188, 162, 168, 120, 50, 150, 177, 105]
after = []

## 메인 코드 부분 ##
print('정렬 전 -->', before)
for _ in range(len(before)) :
    minPos = findMinIdx(before)
    after.append(before[minPos])
    del(before[minPos])
print('정렬 후 -->', after)
```

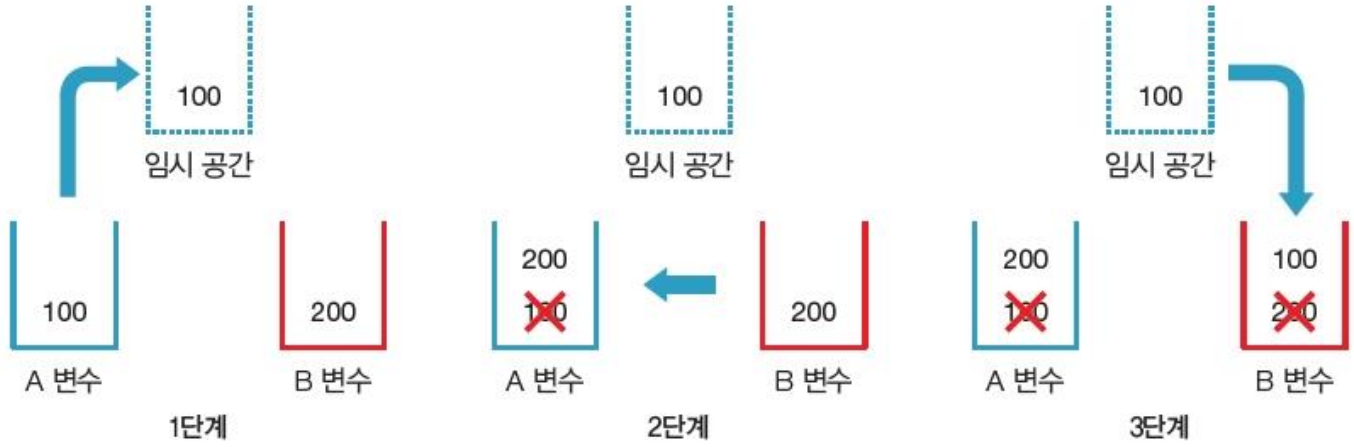
데이터를 탐색하며
가장 작은 값의 인덱스를 찾고
찾아낸 가장 작은 값을 삭제하는 과정을 반복함

실행 결과

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

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




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

또는

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

파이썬은 스택으로 변수를 교환하여 위 방법이 가능

개선된 선택 정렬 구현(데이터 4개를 정렬하는 예)

-> 데이터가 4개이므로 (b)와 같이 총 3회의 사이클이 필요함

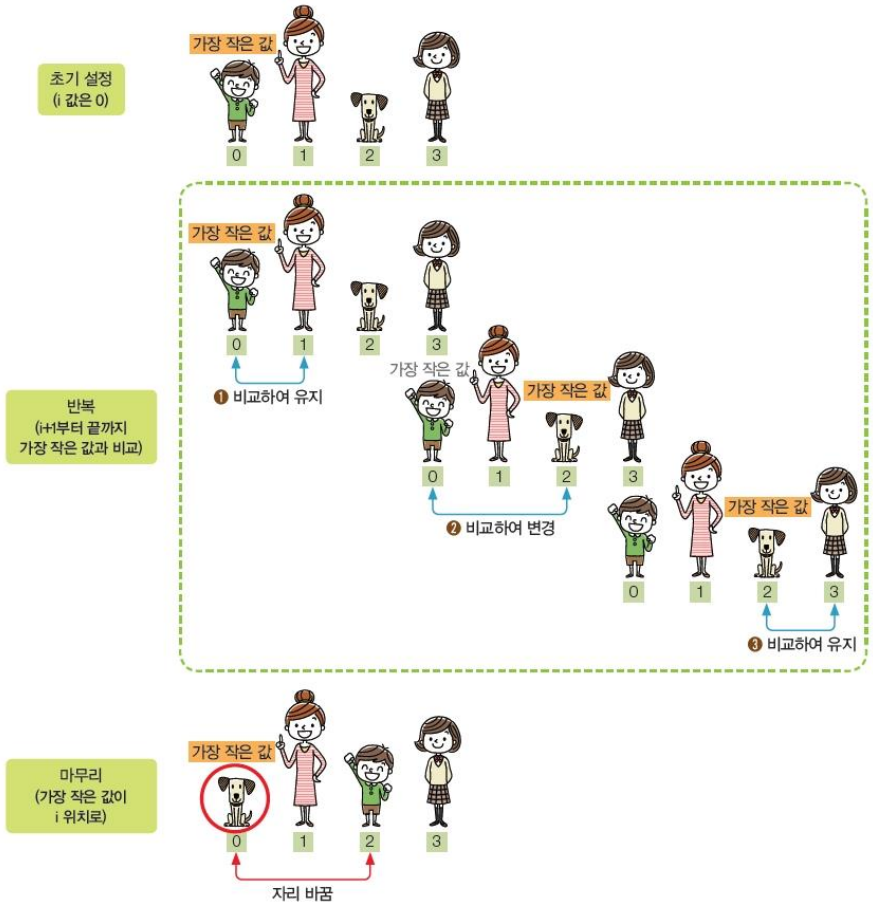
(a) 선택 정렬할 데이터

(b) 3회 사이클

배열을 하나만 사용하고 값을 교환하는 방식
가장 작은 데이터를 왼쪽으로 이동시키며 사이클을 실행

개선된 선택 정렬 구현(데이터 4개를 정렬하는 예)

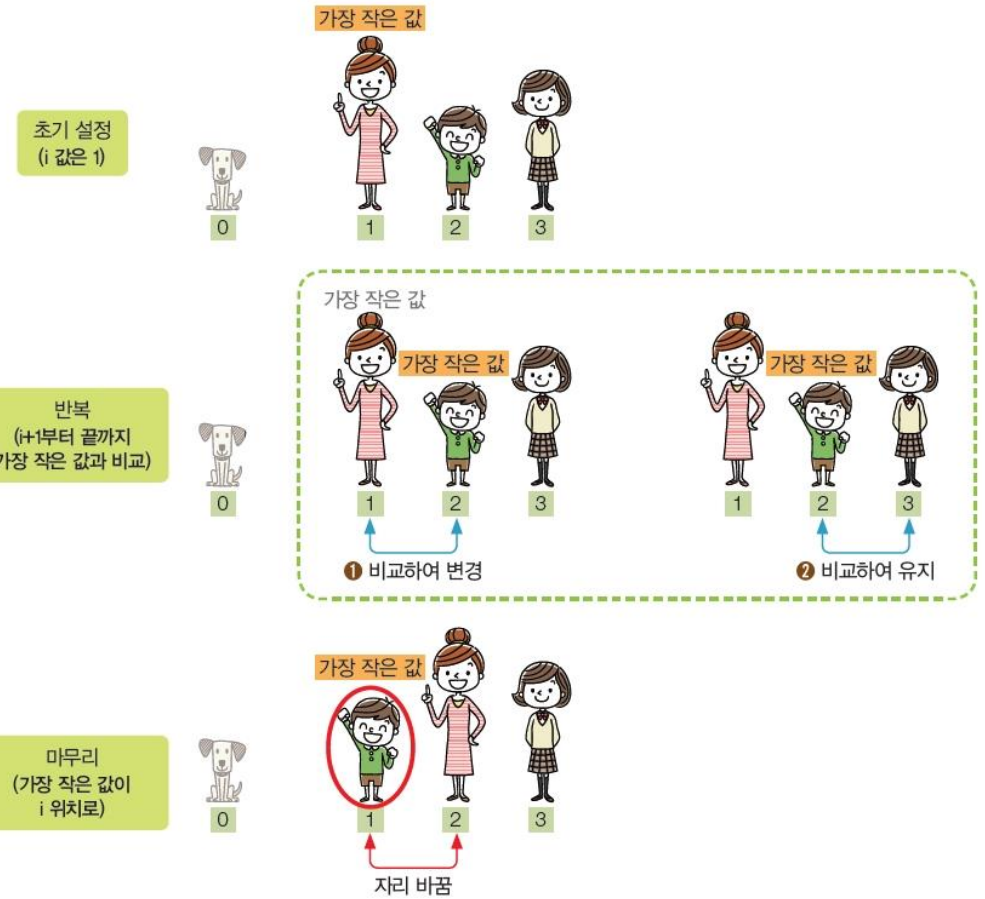
-> 먼저 사이클1 중 맨 앞의 값을 가장 작은 값으로 지정한 후 나머지 값과 비교해서 제일 작은 값을 찾음



사이클 1 :
값을 비교하며
배열의 맨 앞을 가장 작은 값으로 지정

개선된 선택 정렬 구현(데이터 4개를 정렬하는 예)

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



개선된 선택 정렬 구현(데이터 4개를 정렬하는 예)

-> 사이클2에서 찾은 가장 작은 값을 제외한 사이클3 중 맨 앞의 값을 가장 작은 값으로 우선 지정하고 나머지 값들과 비교해서 제일 작은 값을 찾음

정렬 완료

개선된 선택 정렬 구현 코드

```
## 함수 선언 부분 ##
def selectionSort(ary) :
    n = len(ary)
    for i in range(0, n-1) :
        minIdx = i
        for k in range(i+1, n) :
            if (ary[minIdx] > ary[k]) :
                minIdx = k
        tmp = ary[i]
        ary[i] = ary[minIdx]
        ary[minIdx] = tmp

    return ary

## 전역 변수 선언 부분 ##
dataAry = [188, 162, 168, 120, 50, 150, 177, 105]

## 메인 코드 부분 ##
print('정렬 전 -->', dataAry)
dataAry = selectionSort(dataAry)
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]
```

선택 정렬 성능

-> 정렬에서 중요한 사항 중 하나는 정렬을 완료하는 비교 횟수
-> 앞 슬라이드 코드의 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
```

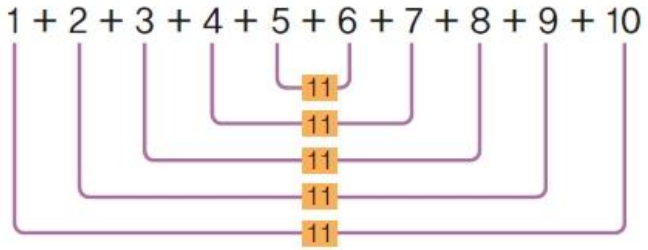
데이터 개수가 4일 때 7행은 이와 같이 반복

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

데이터 개수가 n개라면 이와 같이 계산됨

선택 정렬 성능

-> 비교 횟수는 거꾸로 하면 1+2+3+…+(n-1)번이 되는데
이를 수식으로 유도하기 전에 1부터 10까지 합계를 구하는 방법은 아래와 같음



11 * 5회 = 55	11이 5회 반복되므로 이와 같이 계산됨
(10 + 1) × (10 / 2) = 55	숫자 10을 중심으로 표현

-> 결국 1부터 n까지 합계는 다음 수식과 같음(10 대신에 n을 대입)

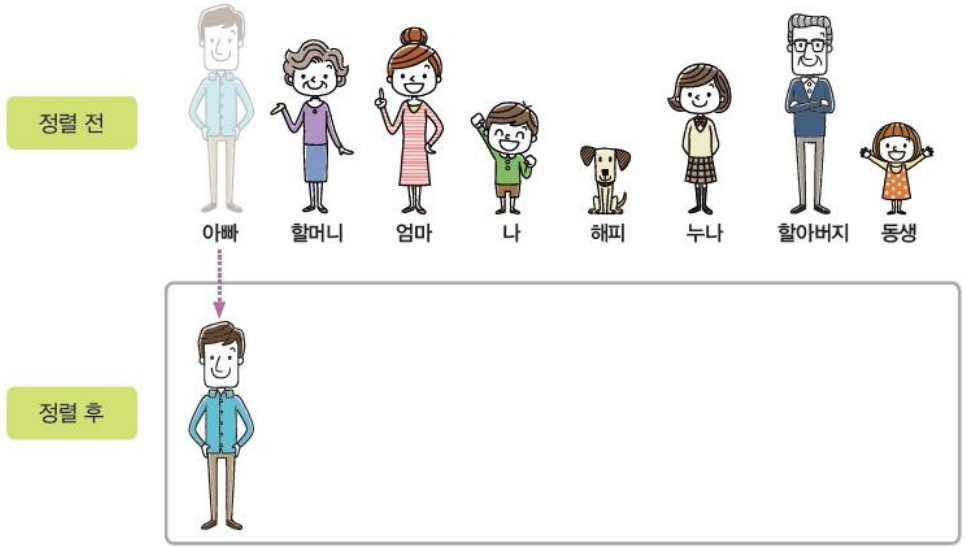
$$(n+1) \times \frac{n}{2} = \frac{(n+1) \times n}{2}$$

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

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

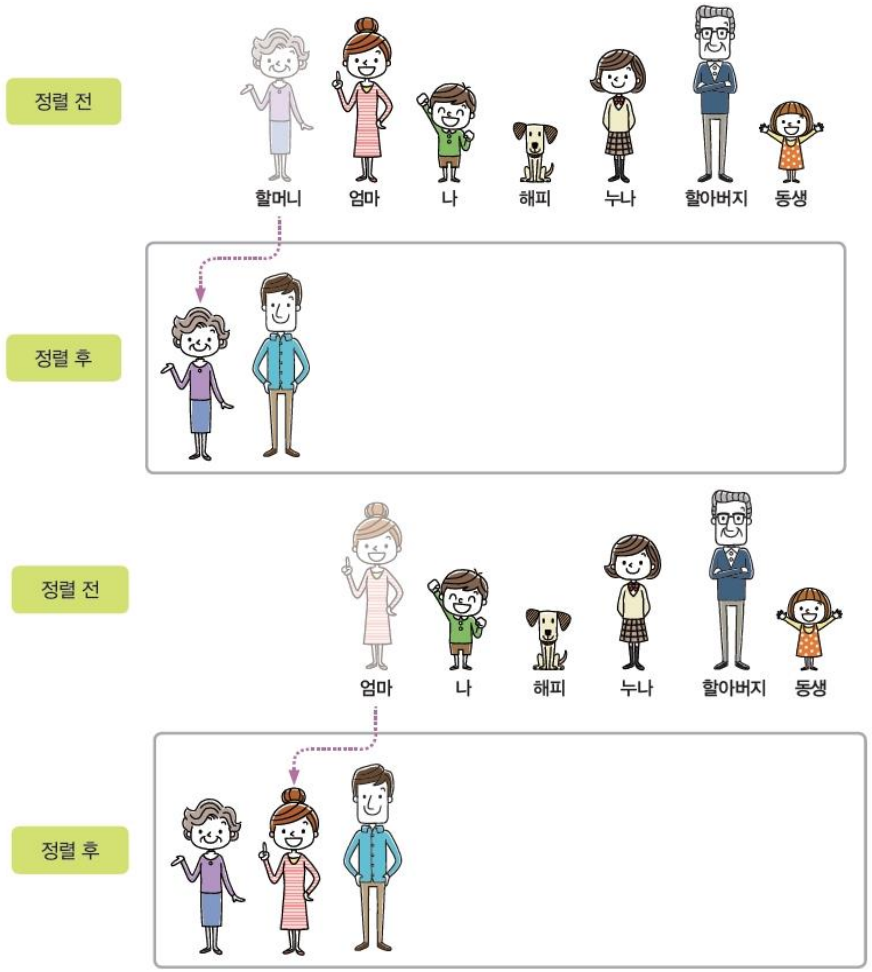
연산 수는 $O(n^2)$
(빅-오 표기법은 최대차수만 고려)

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



1단계 :
가장 앞에 있는 아빠를 일단 줄 앞에 세움

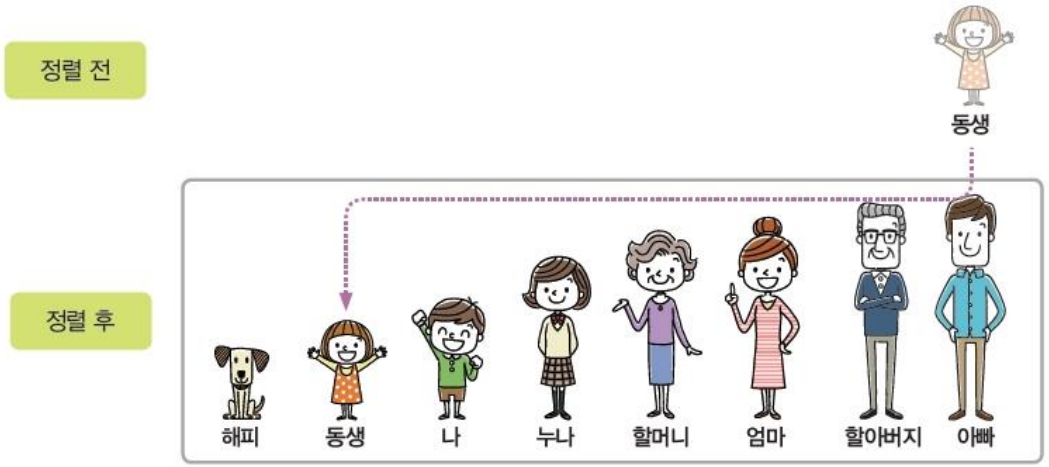
가족을 키 순으로 세우는 삽입 정렬의 예



2단계 :
할머니는 아빠보다 작으므로 아빠 앞에 세움

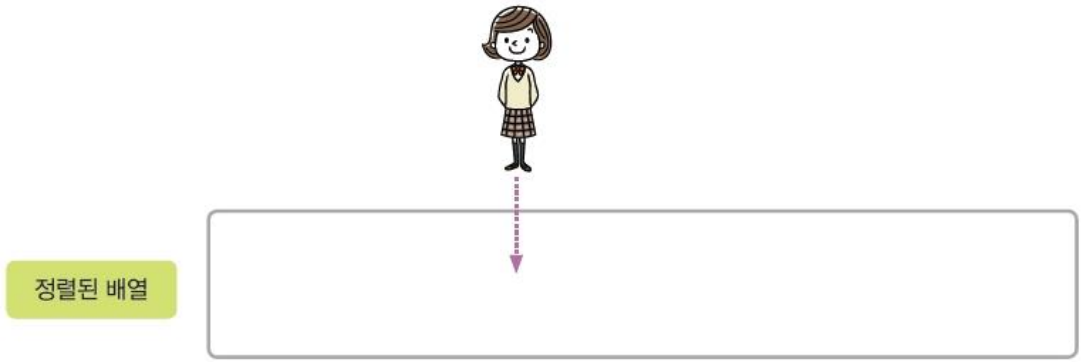
3단계 :
다음으로 엄마를 줄에 세움.
엄마는 할머니보다 크고,
아빠보다 작으므로 그 사이에 세움

가족을 키 순으로 세우는 삽입 정렬의 예

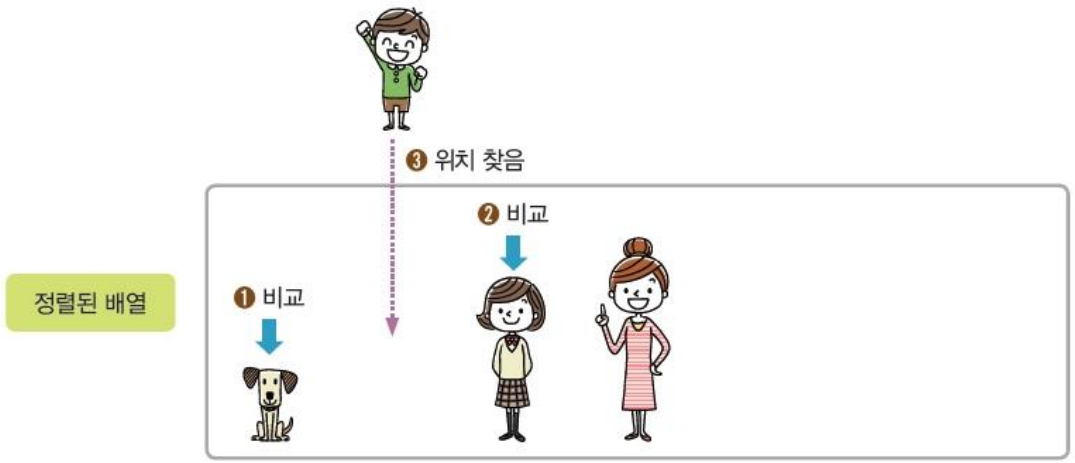


4~8단계 :
같은 방식으로 각 가족을 자신보다 작은 사람과 큰 사람 사이에 세우면 됨
→ 가족을 키 순서대로 오름차순 정렬

삽입 위치를 찾는 방법

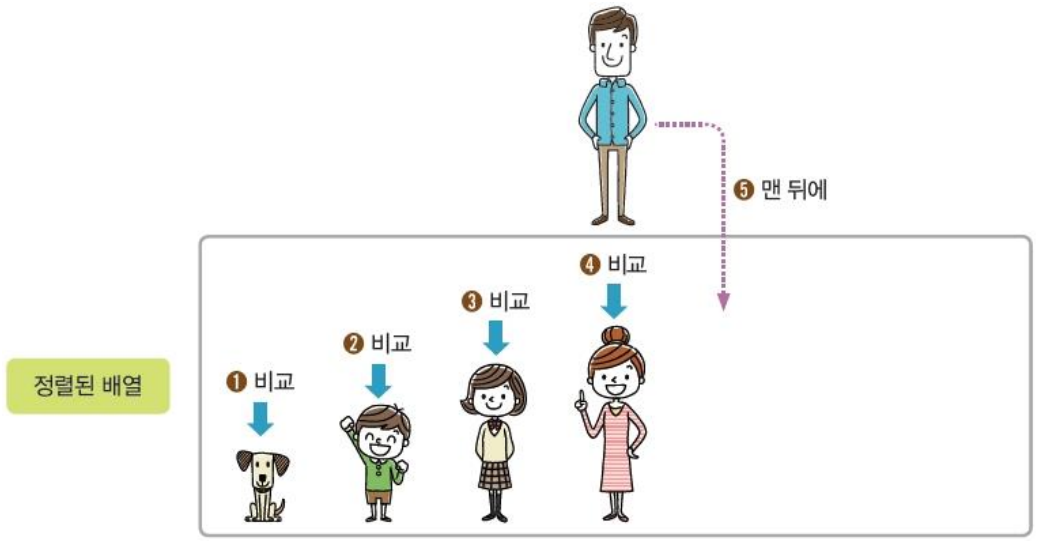


빈 배열일 때는 첫 번째 자리에 삽입함



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

삽입 위치를 찾는 방법



배열에 삽입할 값보다 큰 값이 없을 때는
맨 뒤에 삽입

삽입 위치를 찾는 방법 함수 코드

```
def findInsertIdx(ary, data) :
    findIdx = -1            # 초깃값은 없는 위치로
    for i in range(0, len(ary)) :
        if (ary[i] > data ) :
            findIdx = i
            break
    if findIdx == -1 :          # 큰 값을 못찾음 == 제일 마지막 위치
        return len(ary)
    else :
        return findIdx

testAry = []
insPos = findInsertIdx(testAry, 55)
print('삽입할 위치 -->' , insPos)

testAry = [33, 77, 88]
insPos = findInsertIdx(testAry, 55)
print('삽입할 위치 -->' , insPos)

testAry = [33, 55, 77, 88]
insPos = findInsertIdx(testAry, 100)
print('삽입할 위치 -->' , insPos)
```

실행 결과

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

삽입 정렬 구현

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

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

빈 배열에는 0번째 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]
```

삽입 정렬 구현 코드

```
## 함수 선언 부분 ##
def findInsertIdx(ary, data) :
    findIdx = -1            # 초깃값은 없는 위치로
    for i in range(0, len(ary)) :
        if (ary[i] > data ) :
            findIdx = i
            break
    if findIdx == -1 :          # 큰 값을 못찾음 == 제일 마지막 위치
        return len(ary)
    else :
        return findIdx

## 전역 변수 선언 부분 ##
before = [188, 162, 168, 120, 50, 150, 177, 105]
after = []

## 메인 코드 부분 ##
print('정렬 전 -->', before)
for i in range(len(before)) :
    data = before[i]
    insPos = findInsertIdx(after, data)
    after.insert(insPos, data)
print('정렬 후 -->', after)
```

가장 첫 데이터 입력은 값을 못 찾는 경우이므로
0번째 위치(제일 마지막 위치)에 저장

실행 결과

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

삽입 정렬의 효율적인 구현

-> 배열을 2개 사용하는 것보다 배열 하나에서 데이터를 정렬하는 방식이 더 효율적

(a) 삽입 정렬할 데이터

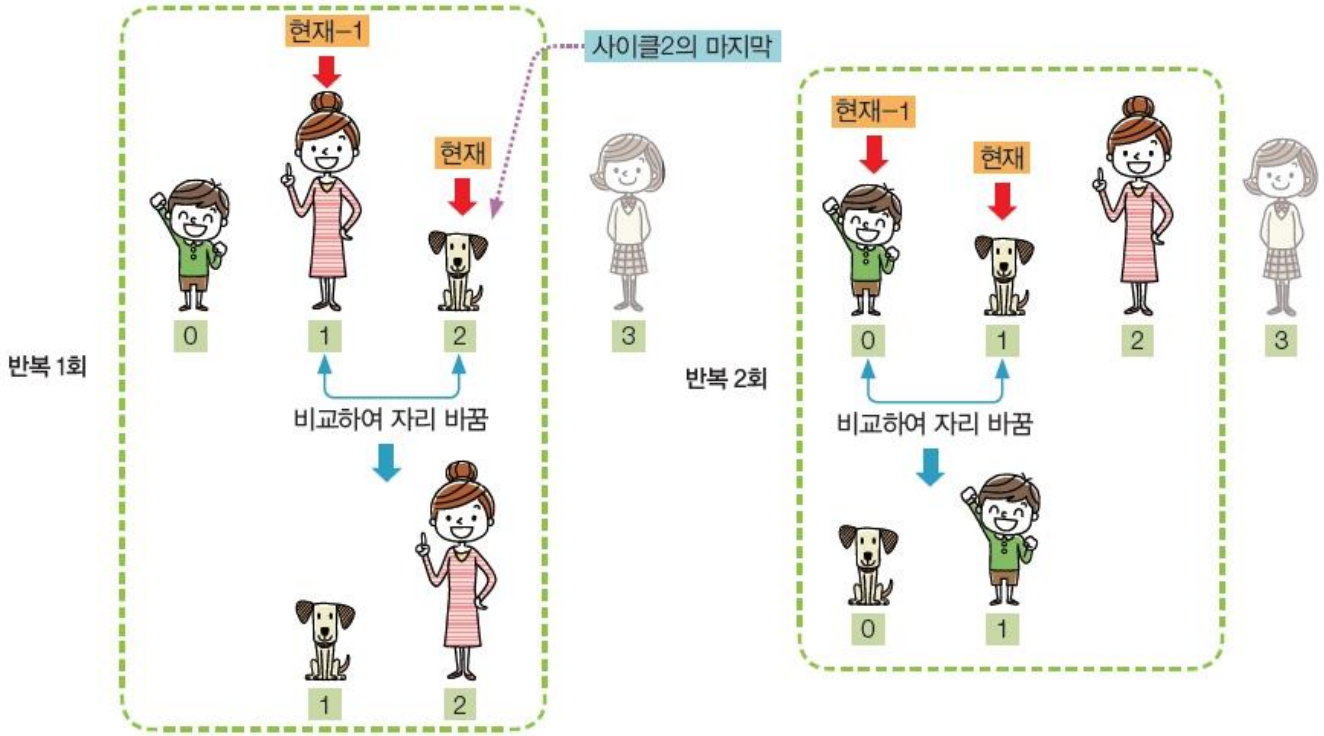
(b) 3회 사이클

데이터가 4개인 경우 3사이클이 필요

삽입 정렬의 효율적인 구현

사이클 1 :
사이클 1의 마지막 데이터를 현재로 두고
두 데이터를 비교해서 작은 것을 앞으로 가져옴

삽입 정렬의 효율적인 구현



사이클 2 :
사이클 중 마지막 2개부터 각 쌍을
비교해서 작은 것을 앞으로 가져옴

삽입 정렬의 효율적인 구현

사이클 3 :
사이클 중 마지막 2개부터 각 쌍을 비교해서 작은 것을 앞으로 가져옴

정렬 완료
(이게 최선일까?)

삽입 정렬의 효율적인 구현 소스 코드

```
## 함수 선언 부분 ##
def insertionSort(ary) :
    n = len(ary)
    for end in range(1, n) :
        for cur in range(end, 0, -1) :
            if ( ary[cur-1] > ary[cur] ) :
                ary[cur-1], ary[cur] = ary[cur], ary[cur-1]
    return ary

## 전역 변수 선언 부분 ##
dataAry = [188, 162, 168, 120, 50, 150, 177, 105]

## 메인 코드 부분 ##
print('정렬 전 -->', dataAry)
dataAry = insertionSort(dataAry)
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]
```

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

정렬 기본 알고리즘 응용

1차원 배열의 중앙값 계산

-> 평균값 또는 중앙값을 활용하여 일반적인 데이터 현황을 분석
-> 그러나 평균값은 비정상 값이 포함된 경우 데이터에 혼란이 발생

7	5	11	6	9	80000	10	6	15	12

데이터 값 중에서 비정상적인 수치가 섞여 있는 예, 평균값이 매우 높아지게 됨

-> 중앙값은 데이터를 일렬로 정렬해서 나열한 후 나열된 숫자의 가운데에 위치하는 값을 대푯값으로 하는 방법
-> 비정상 값이 포함되더라도 전체 데이터에서 중간의 위치를 선택하므로 문제 없음

5	6	6	7	9	10	11	12	15	80000

1차원 배열의 중앙값 계산 코드

```
def selectionSort(ary) :
    n = len(ary)
    for i in range(0, n-1) :
        minIdx = i
        for k in range(i+1, n) :
            if (ary[minIdx] > ary[k] ) :
                minIdx = k
        tmp = ary[i]
        ary[i] = ary[minIdx]
        ary[minIdx] = tmp

    return ary

## 전역 변수 선언 부분 ##
moneyAry = [7, 5, 11, 6, 9, 80000, 10, 6, 15, 12]

## 메인 코드 부분 ##
```

```
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
```

위와 같이 출력되도록
사려진 코드 부분을 완성하자

파일 이름의 정렬 출력

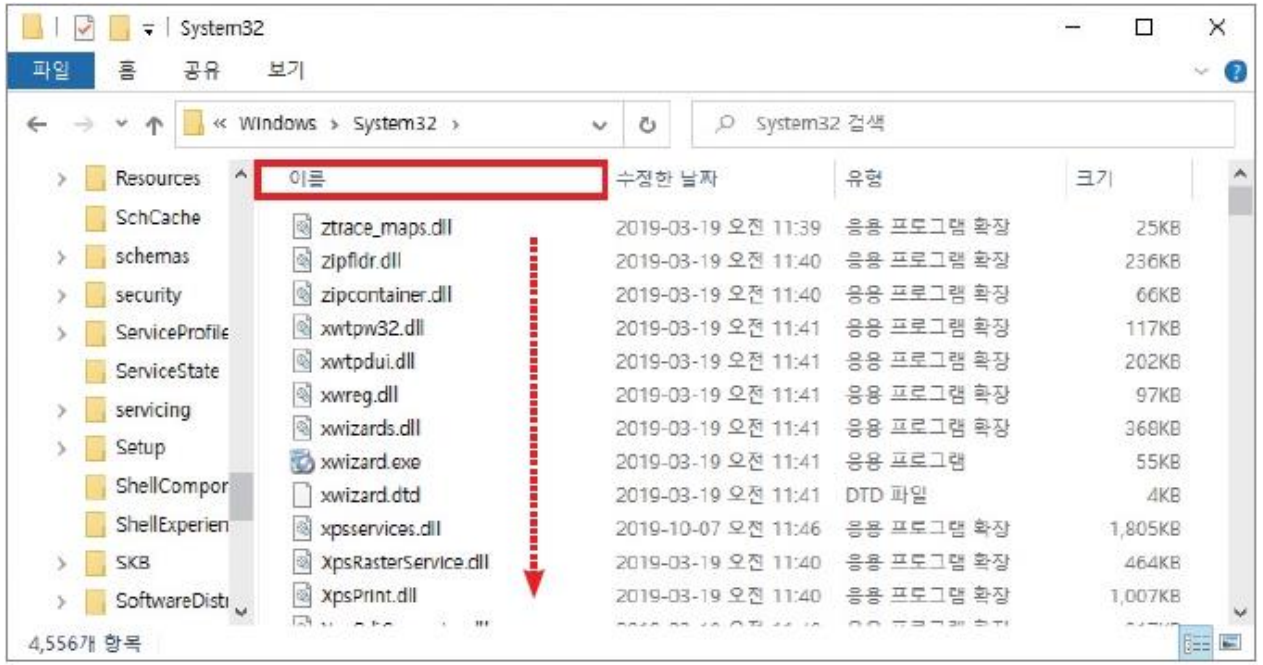
-> 지정된 폴더에서 하위 폴더를 포함한 파일 목록 추출

```
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))
```

C: \ Windows \ System32 폴더의 파일을 배열에 저장하는 기능을 하는 코드
os.walk : 매개변수로 전달한 폴더를 탐색하면서 경로, 경로의 폴더들, 경로의 파일들을 튜플로 리턴함

파일 이름의 정렬 출력

-> 파일은 '파일명.확장명'으로 구분하고 다양한 방법으로 정렬 가능



파일명을 내림차순으로 정렬한 상태

파일 이름의 내림차순 정렬 출력 코드

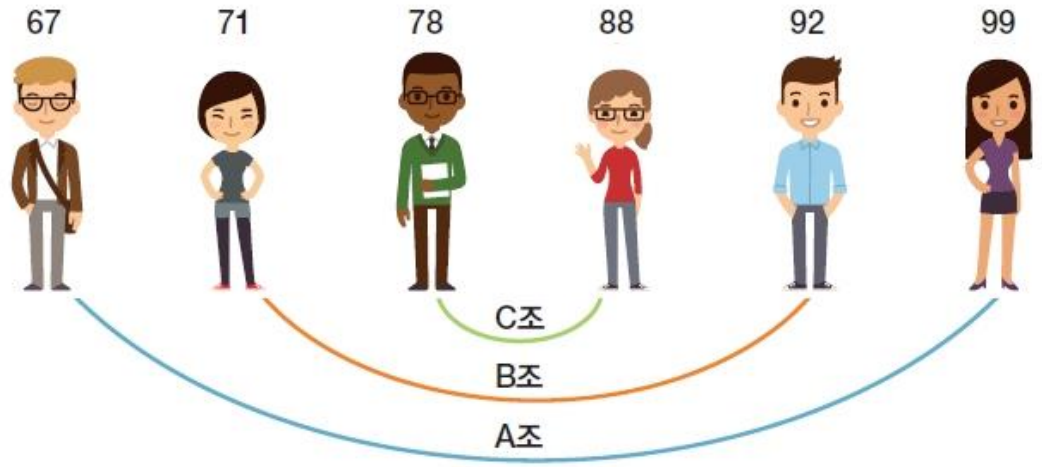
```
import os

## 함수 선언 부분 ##
def makeFileList(folderName) :
    fnameAry = []
    for dirName, subDirList, fnames in os.walk(folderName):
        for fname in fnames:
            fnameAry.append(fname)
    return fnameAry

def insertionSort(ary) :
    [                                    ]

## 전역 변수 선언 부분 ##
fileAry = []

## 메인 코드 부분 ##
fileAry = makeFileList('C:/Program Files/Common Files')
fileAry = insertionSort(fileAry)
print('파일명 역순 정렬 -->', fileAry)
```

파일명 역순 정렬 --> ['zh-phonetic.xml', 'zh-dayi.xml', 'zh-changjei.xml', 'xmsrv_xl.dll', 'xmlrwbin_xl.dll', 'xmlrwbin.dll', 'xmlrw_xl.dll', 'xmlrw.dll', 'xlsrvintl.dll', 'wab32res.dll.mui', 'wab32res.dll.mui', 'wab32res.dll', 'wab32.dll', 'vstoee90.tlb', 'vstoee100.tlb', 'vstoee.dll', 'vsjitdebuggerps.dll',

정렬이 되도록 삽입 정렬 코드를 완성해 넣고 실행해보자

응용 예제 1

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

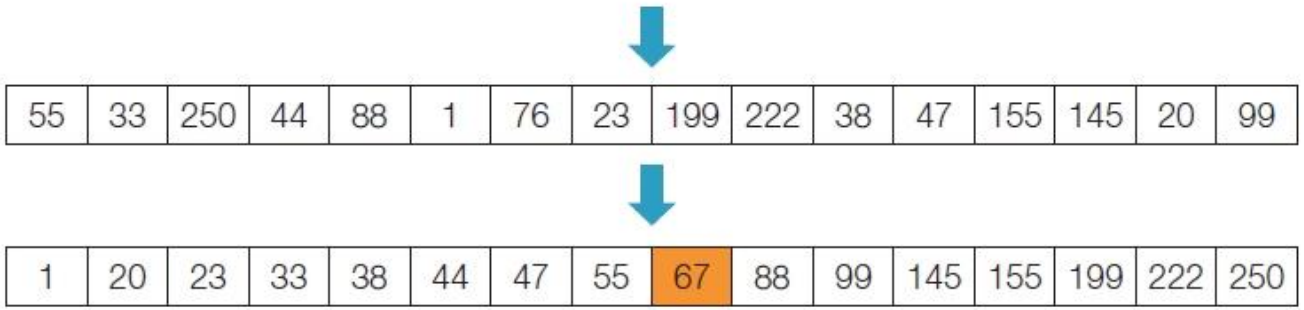
```
Python
File  Edit  Shell  Debug  Options  Window  Help
========== RESTART: C:₩CookData₩Ex11-01.py ==========
정렬 전 —> [['선미', 88], ['초아', 99], ['화사', 71], ['영탁', 78], ['영웅', 67], ['민호', 92]]
정렬 후 —> [['영웅', 67], ['화사', 71], ['영탁', 78], ['선미', 88], ['민호', 92], ['초아', 99]]
## 성적별 조 편성표 ##
영웅 : 초아
화사 : 민호
영탁 : 선미
>>>
Ln: 30  Col: 4
```

응용 예제 1 코드

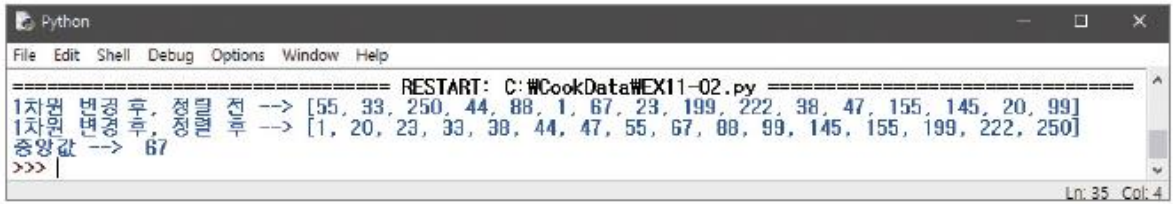
응용 예제 2

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



실행 결과



응용 예제 2 코드

정렬 방법은 선택, 삽입 정렬 중
아무거나 선택해서 구현해도 됨

다음 강의 예고

정렬 고급

- 정렬 고급 기본
- 고급 정렬 알고리즘의 원리와 구현
- 고급 정렬 알고리즘의 응용

Q&A

감사합니다.