CHAPTER.22

기본 정렬 알고리즘의 원리와 구현 및 응용

학습 내용

[1] 기본 정렬 알고리즘의 원리와 구현 (삽입 정렬)

[2] 기본 정렬 알고리즘의 응용

학습 목표

- 기본적인 정렬 방식에 대해 설명할 수 있다.
- 정렬을 활용한 응용 프로그램을 작성할 수 있다.

01 기본 정렬 알고리즘의 원리와 구현

1) 삽입 정렬

삽입 정렬의 개념

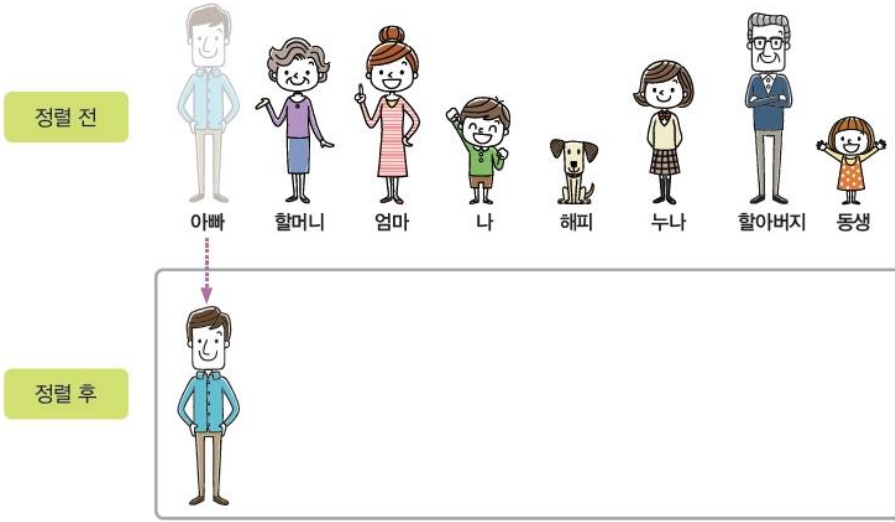
기존 데이터 중에서 자신의 위치를 찾아 데이터를 삽입하는 정렬

- 가족을 키 순으로 세우는 예에 삽입 정렬 적용

0 삽입 정렬 전 초기 상태

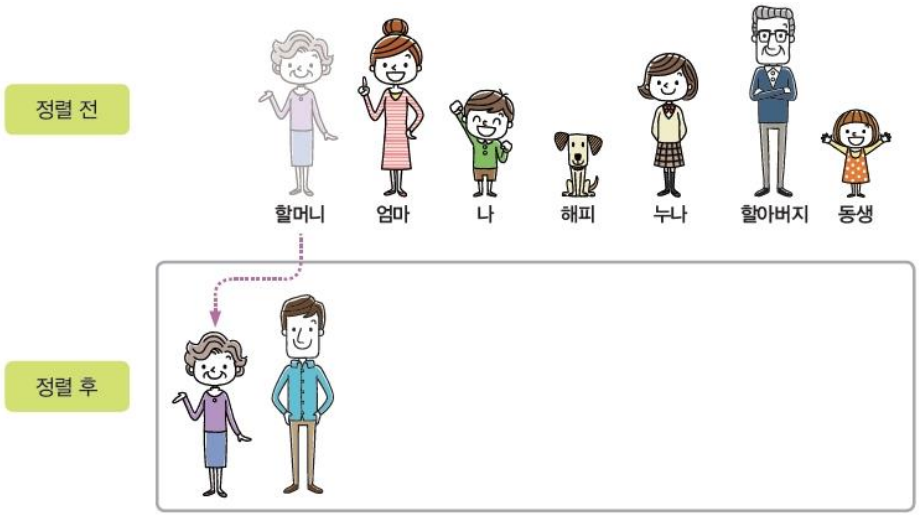
삽입 정렬의 개념

1 가장 앞에 있는 아빠를 일단 줄에 세움



삽입 정렬의 개념

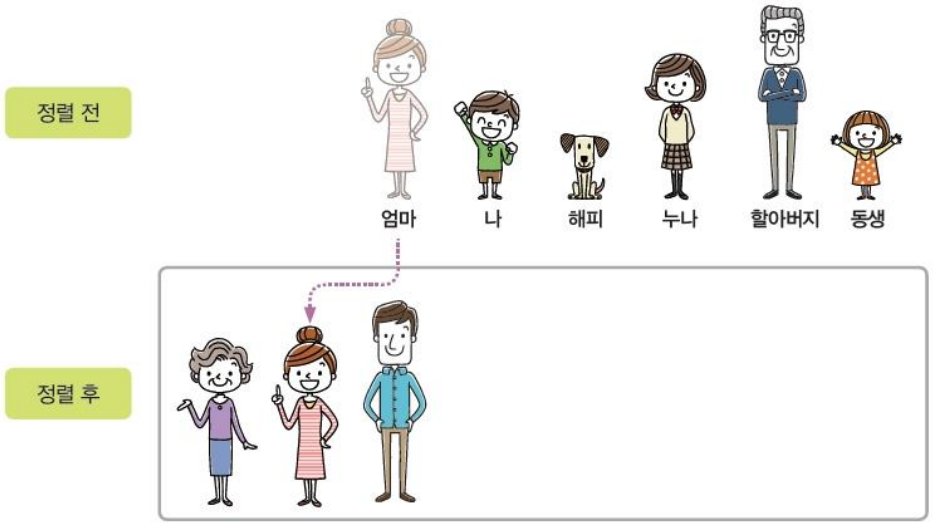
2 할머니는 아빠보다 작으므로 아빠 앞에 세움



삽입 정렬의 개념

3 다음으로 엄마를 줄에 세움

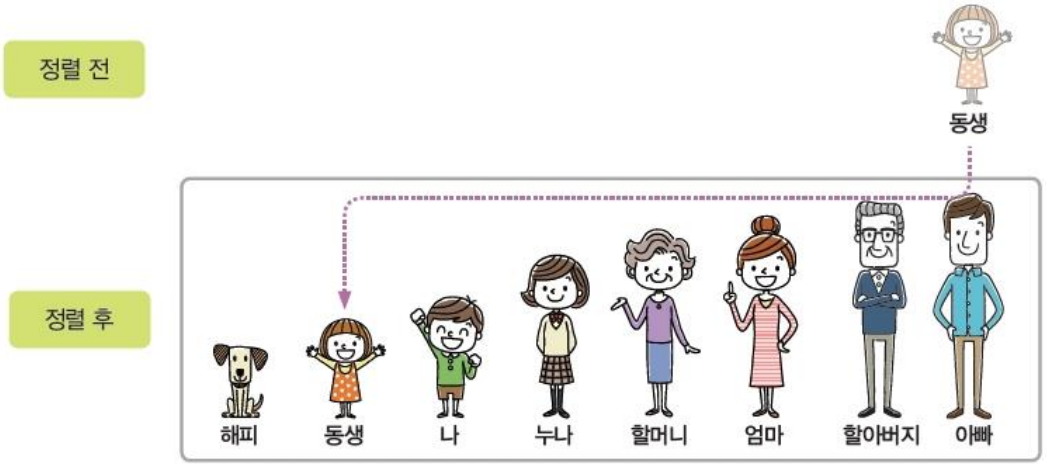
- 엄마는 할머니보다 크고, 아빠보다 작으므로 그 사이에 세움



삽입 정렬의 개념

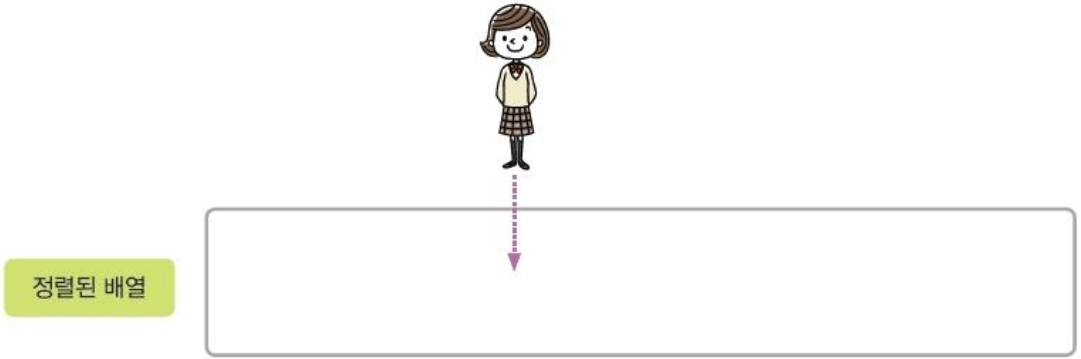
4 같은 방식으로 각 가족을 자신보다 작은 사람과 큰 사람 사이에 세우면 됨

- 가족을 키 순서대로 오름차순 정렬



삽입 위치를 찾는 방법

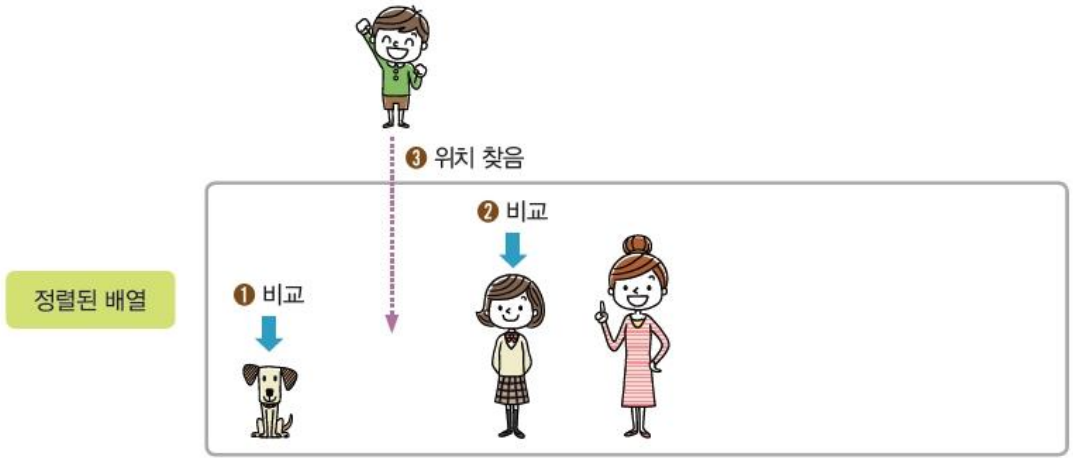
1 빈 배열일 때는 첫 번째 자리에 삽입함



삽입 위치를 찾는 방법

2 배열에 삽입할 값보다 큰 값이 있을 때

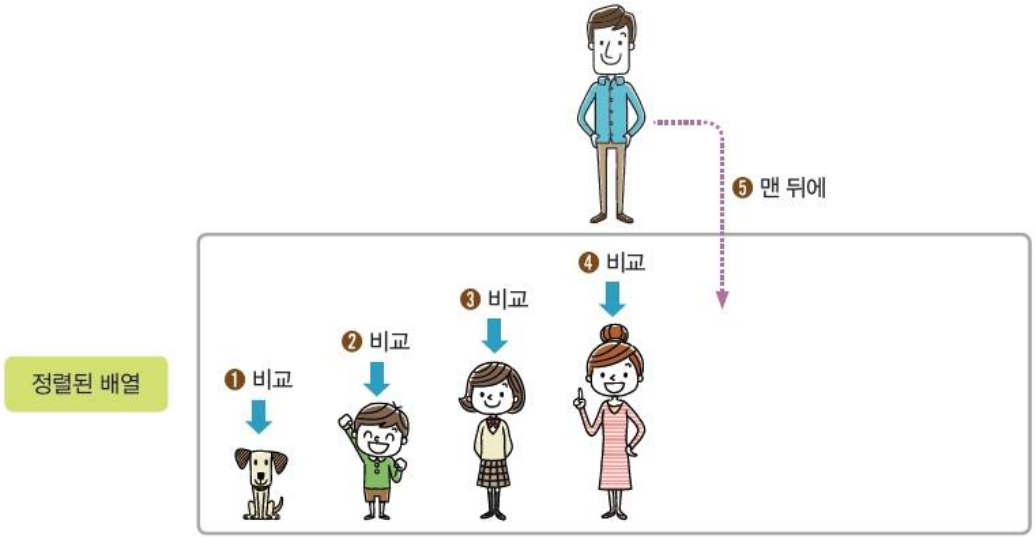
- 처음부터 비교해 가면서 자신보다 큰 값을 만나면 그 값 바로 앞에 삽입



삽입 위치를 찾는 방법

3 배열에 삽입할 값보다 큰 값이 없을 때

- 맨 뒤에 삽입



삽입 위치를 찾는 방법

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
```

삽입 위치를 찾는 방법

배열에서 자신이 삽입될 위치를 찾는 함수

```
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
```

삽입 정렬 구현

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

- **빈 배열에는 X번째 위치에 값을 삽입**

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

실행 결과

```
[55]
```

삽입 정렬 구현

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

- 1번째 위치에 55를 삽입

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

실행 결과

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

삽입 정렬 구현

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

- 배열의 맨 뒤에 값을 삽입
- 4번째 위치에 100을 삽입

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

실행 결과

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

삽입 정렬 구현

삽입 정렬의 구현

```
## 클래스와 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 못 찾음 == 제일 마지막 위치
        return len(ary)
    else :
        return findIdx

```

삽입 정렬 구현

삽입 정렬의 구현

```
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]
```

삽입 정렬 실습

앞쪽의 소스를 수정해서 랜덤하게 0~200 사이의 숫자 10개를 생성한 후, 내림차순으로 정렬하도록 코드를 작성하자.

(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]
```

삽입 정렬 실습

삽입 정렬의 효율적인 구현

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

간단히 데이터 4개를 정렬한 예(총 3회의 사이클이 필요)

- 삽입 정렬 전 초기 상태

삽입 정렬할 데이터

3회 사이클

삽입 정렬의 효율적인 구현

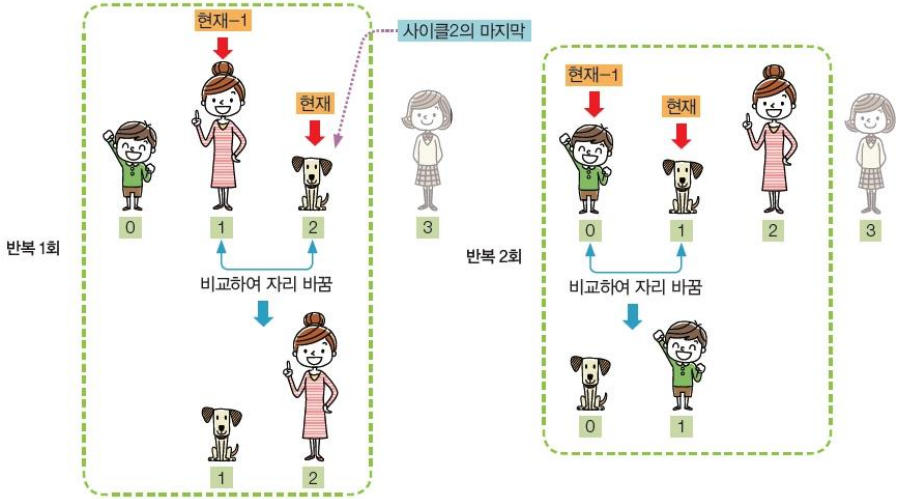
1 먼저 사이클1의 마지막을 현재로 두고
두 데이터를 비교해서 작은 것을 앞으로 가져옴

- 삽입 정렬 예 – 사이클1

삽입 정렬의 효율적인 구현

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

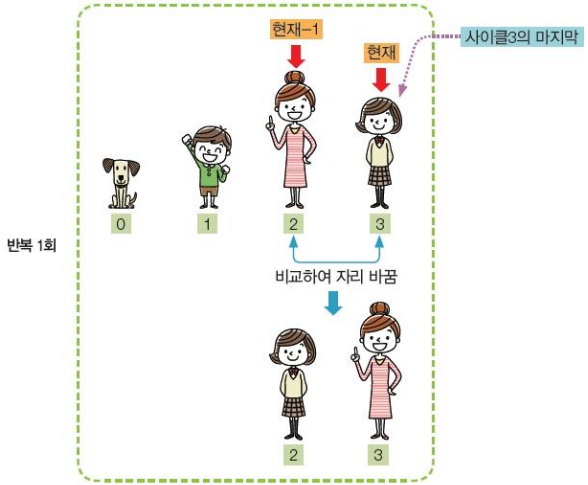
▪ 삽입 정렬 예 - 사이클2



삽입 정렬의 효율적인 구현

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

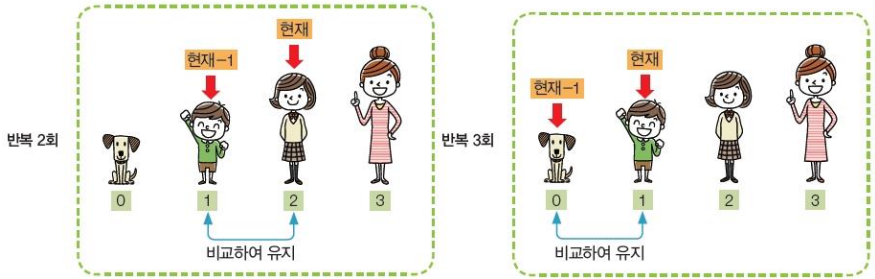
- 삽입 정렬 예 – 사이클3



삽입 정렬의 효율적인 구현

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

- 삽입 정렬 예 – 사이클3



삽입 정렬의 효율적인 구현

4 모든 사이클 완료 후 정렬된 결과

- 정렬 후 데이터

삽입 정렬의 효율적인 구현

삽입 정렬의 효율적인 구현

```
## 클래스와 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]

```

삽입 정렬의 효율적인 구현

삽입 정렬의 효율적인 구현

```
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]
```

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

02

기본 정렬 알고리즘의 응용

1) 1차원 배열의 중앙값 계산
2) 파일 이름의 정렬 출력
3) 기본 정렬 알고리즘의 응용 실습

1] 1차원 배열의 중앙값 계산

평균값 계산

평균값

- 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

1] 1차원 배열의 중앙값 계산

중앙값 계산

중앙값

- 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

중앙값 계산

중앙값 계산

```
## 클래스와 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

```

중앙값 계산

중앙값 계산

```
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
```

2] 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
```

2] 파일 이름의 정렬 출력

지정된 폴더에서 하위 폴더를 포함한 파일 목록 추출

파일 목록

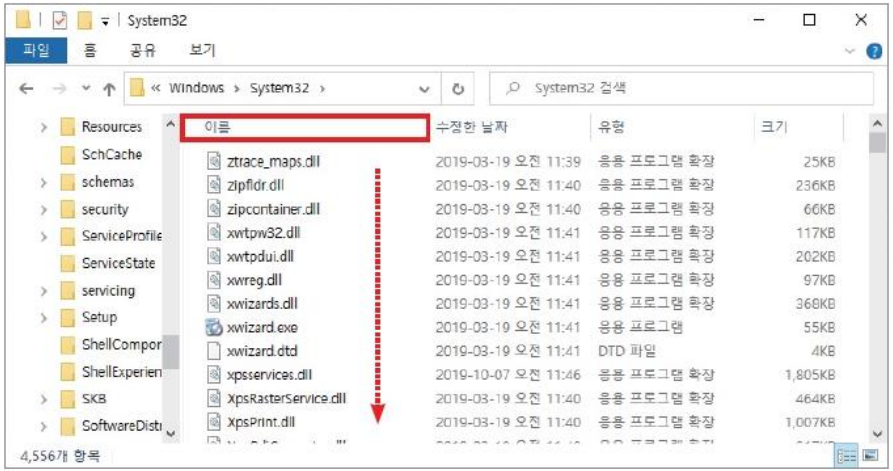
```
['accessibilitycpl.dll', '12520437.cpx', '12520850.cpx', 'aadtb.dll', 'BitLockerCsp.
dll', 'AarSvc.dll', 'AboveLockAppHost.dll', …
```

2] 파일 이름의 정렬 출력

파일 목록 정렬

파일은 '파일명.확장자명'으로 구분

- 파일명 및 확장자명으로 분리해서 정렬하는 경우
- 파일 탐색기에서 파일명을 내림차순으로 정리한 상태



파일 목록 정렬

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]) :
```

파일 목록 정렬

파일명으로 내림차순 정렬

```
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)
```

파일 목록 정렬

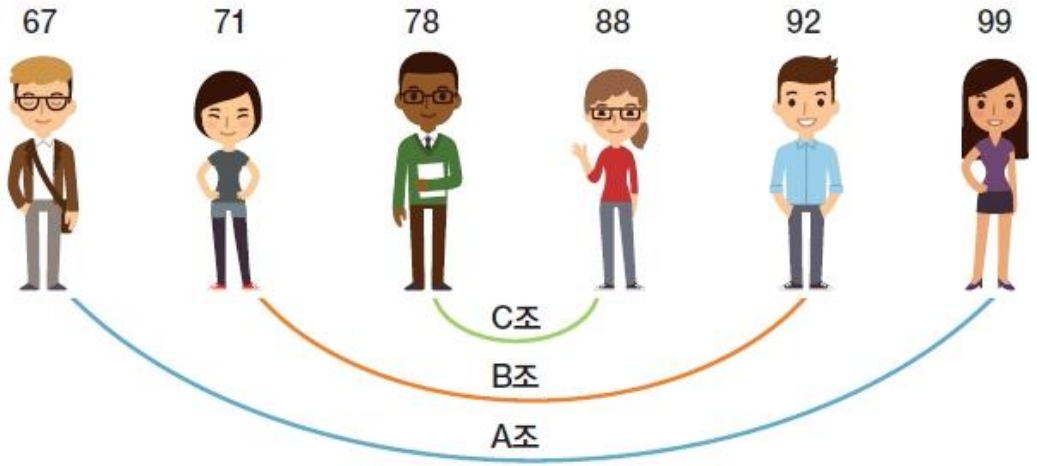
파일명으로 내림차순 정렬

실행 결과

```
파일명 역순 정렬 --> ['zh-phonetic.xml', 'zh-dayi.xml', 'zh-changjei.xml', 'wab32res.dll.mui', 'wab32res.dll', 'wab32.dll', 'vstoee90.tlb', 'vstoee100.tlb', 'vstoee.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)
```

기본 정렬 알고리즘의 응용 실습

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



기본 정렬 알고리즘의 응용 실습

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

실행 결과

기본 정렬 알고리즘의 응용 실습

학습평가

Q1 Q2

Q1 삽입 정렬의 성능에 대한 빅-오 표기법은 다음 중 무엇인가?

1. $O(n)$
2. $O(2n)$
3. $O(n^2)$
4. $O(n^3)$

학습평가

Q1 삽입 정렬의 성능에 대한 빅-오 표기법은 다음 중 무엇인가?

1. $O(n)$
2. $O(2n)$
3. ✔ $O(n^2)$
4. $O(n^3)$

정 답 3 $O(n^2)$

해 설 선택 정렬과 삽입 정렬의 성능은 $O(n^2)$입니다.

학습평가

Q1 Q2

Q2 삽입 정렬에 대한 설명으로 거리가 먼 것은?

1. 기존 데이터 중에서 자신의 위치를 찾아 데이터를 삽입하는 정렬이다.
2. 키 순서, 이름 순서, 몸무게 순서 등을 정렬할 때 사용할 수 있다.
3. 정렬은 오름차순과 내림차순으로 정렬할 수 있다.
4. 개념은 어렵지만 속도가 가장 빠른 정렬 중 하나이다.

학습평가

Q1 Q2

Q2 삽입 정렬에 대한 설명으로 거리가 먼 것은?

1 기존 데이터 중에서 자신의 위치를 찾아 데이터를 삽입하는 정렬이다.

2 키 순서, 이름 순서, 몸무게 순서 등을 정렬할 때 사용할 수 있다.

3 정렬은 오름차순과 내림차순으로 정렬할 수 있다.

✓ 4 개념은 어렵지만 속도가 가장 빠른 정렬 중 하나이다.

정 답 4 개념은 어렵지만 속도가 가장 빠른 정렬 중 하나이다.

해 설 입력 개수가 커질수록 기하급수적으로 비교 횟수(또는 연산 횟수)가 늘어나기에 성능이 좋지 않은 알고리즘입니다.

기본 정렬 알고리즘의 원리와 구현

삽입 정렬

- 삽입 정렬의 개념 : 기존 데이터 중에서 자신의 위치를 찾아 데이터를 삽입하는 정렬
- 삽입 정렬의 효율적인 구현 : 배열을 2개 사용하는 것보다 배열 하나에서 데이터를 정렬하는 방식이 더 효율적

삽입 정렬의 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

기본 정렬 알고리즘의 응용

1차원 배열의 중앙값 계산

- 평균값 : 전체 데이터 값을 합친 후 개수로 나누는 것
- 중앙값 : 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