

*Recursion and Iteration : Sorting*

# #05. 재귀와 반복 : 정렬

김현하

한양대학교 ERICA 소프트웨어학부

2021.10.12.  
2021년도 2학기

# 목차

- 시퀀스
  - 리스트, 튜플, 문자열, 정수범위, 시퀀스 연산, **for** 루프
- 리스트 정렬
  - 선택정렬, 삽입정렬, 합병정렬, 퀵정렬

# 정렬 sorting

- 정렬
  - 순서를 매기는 기준을 정해놓고  
그 순서에 맞게 데이터를 차례로 나열하는 문제
- Python의 리스트<sup>list</sup>
  - 데이터 원소를 쉼표로 구분하여 나열하고 **대괄호**로 둘러쌘
    - **[3, 5, 4, 2]**
    - **["컴퓨터과학", "대학교", "소프트웨어", "인공지능"]**
  - <리스트>.sort() 메소드로 <리스트>의 정렬 가능

# 정렬 sorting

- [[Python 인터프리터](#)]

```
numbers = [3, 5, 4, 2]
```

```
print("before sorting: ", numbers)
```

```
numbers.sort()
```

```
print("after sorting: ", numbers)
```

```
words = ["컴퓨터과학", "대학교", "소프트웨어", "인공지능"]
```

```
print("before sorting: ", words)
```

```
words.sort()
```

```
print("after sorting: ", words)
```

# 정렬 sorting

- 오늘 배울 정렬 함수의 입출력 요구사항
  - 입력(파라미터)
    - 순서를 매길 수 있는 원소로 구성된 리스트  $s$
  - 출력(리턴)
    - $s$ 를 감소하지 않는 순으로 정렬한 리스트

크기가 같은 원소가 둘 이상 있을 수 있음

# 시퀀스

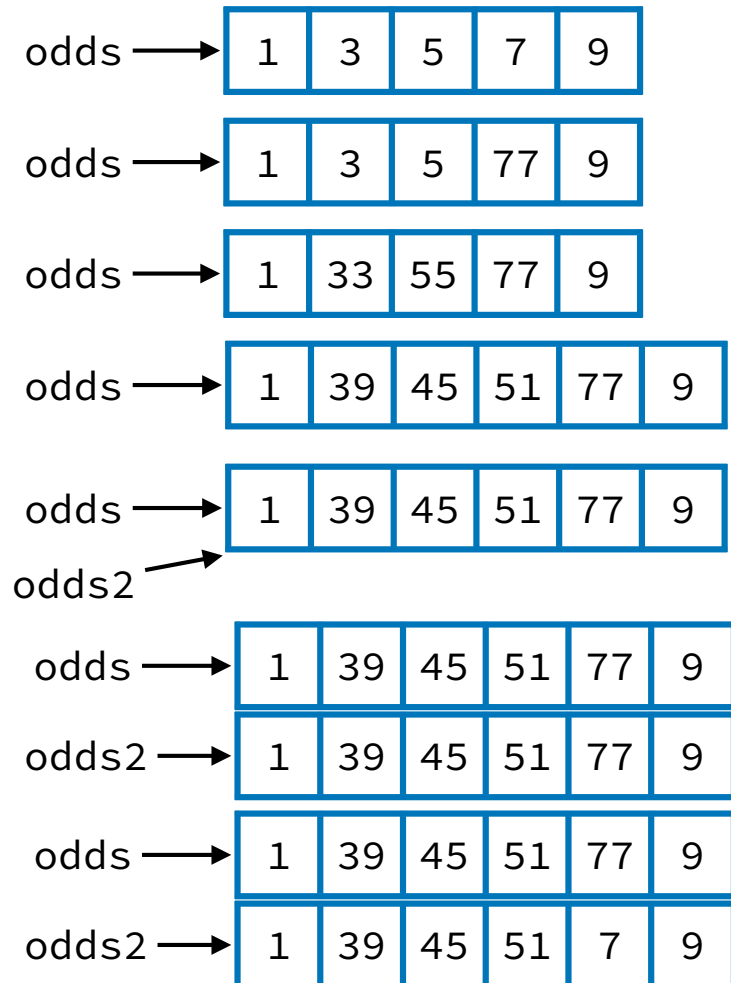
# 시퀀스

- 시퀀스sequence
  - 여러 개의 데이터를 한 줄로 세워서 모아놓은 데이터 구조의 통칭
  - 원소의 위치번호인 인덱스index를 사용해서 접근 (일부는 수정도) 가능
  - Python 에서 제공하는 시퀀스

시퀀스 종류	수정가능여부	정렬연산 제공
리스트list	수정가능	제공
튜플tuple	수정불가능	제공하지 않음
문자열string	수정불가능	제공하지 않음
정수범위range	수정불가능	제공하지 않음

# 리스트

- `odds = [1, 3, 5, 7, 9]`
- `odds[3] = 77`
- `odds[1:3] = [33, 55]`
- `odds[1:3] = [39, 45, 51]`
- `odds2 = odds`
- `odds2 = odds[:]`
- `odds2[4] = 7`





# 튜플

- 원소를 심표로 구분하여 나열하고 전체를 괄호로 둘러싸아 표현, 수정불가능

- [[Python 인터프리터](#)]

- `t = ('컴퓨터과학', '인공지능', 2021)`

- `t`

- `t[2]`

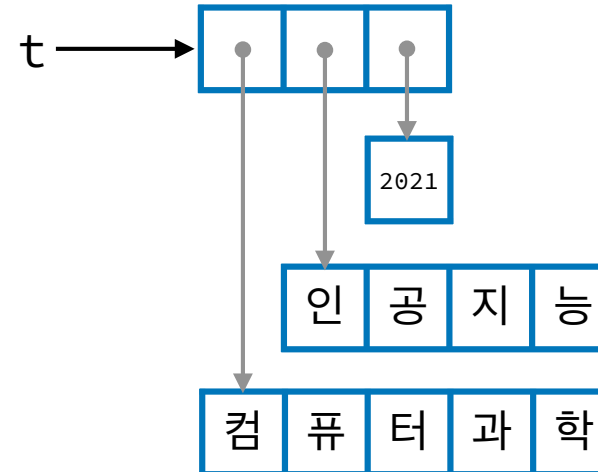
- `t[1:]`

- `t[:2]`

- `t[2] = '깡'`

- `("alone",)` # `("alone")` # 원소가 하나인 튜플

- `()` # 빈 튜플



# 문자열

- 수정불가능

- [[Python 인터프리터](#)]

- `s = '컴퓨터과학'`

- `s[3] = '공'`

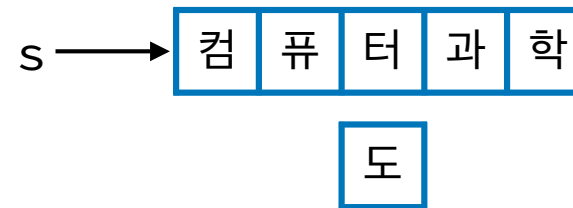
- `s + '도'`

- `s`

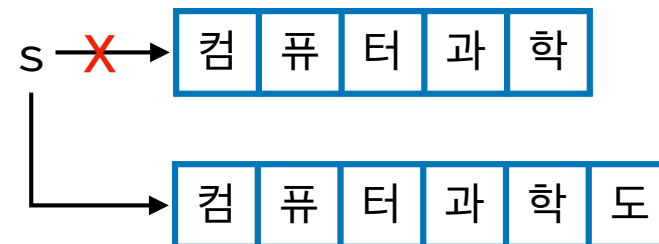
- `s = s + '도' # 지정문`

- `s`

지정문 실행전



지정문 실행후



# 정수범위

- 정수를 일정 간격으로 나열한 시퀀스
- `range(n)`
  - 정수 0부터 n-1까지를 간격 1로 나열한 정수범위 시퀀스
  - `range(5)` : 0,1,2,3,4
- `range(m, n)`
  - 정수 m부터 n-1까지를 간격 1로 나열한 정수범위 시퀀스
  - `range(3, 10)` : 3,4,5,6,7,8,9
- `range(m, n, k)`
  - 정수 m부터 n-1까지를 간격 k로 나열한 정수범위 시퀀스
  - `range(3,11,2)` : 3,5,7,9
  - `range(10,3,-1)` : 10,9,8,7,6,5,4

# 정수범위

- `in` 은 모든 시퀀스가 공유하는 중위표기 논리연산자로 특정 원소가 시퀀스에 있는지 확인할 수 있음

- [[Python 인터프리터](#)]

- `r = range(3,11,2) # 3,5,7,9`

- `5 in r`

- `8 in r`

- `11 in r`

- `r[2] = 3`

# 시퀀스 연산

연산	의미
<code>x in s</code>	x가 s에 있으면 <code>True</code> , 없으면 <code>False</code>
<code>x not in s</code>	x가 s에 없으면 <code>True</code> , 있으면 <code>False</code>
<code>s[i]</code>	s에서 i 위치에 있는 원소
<code>s[i:j]</code>	위치 i에서 위치 j까지 시퀀스 조각 (i 원소 포함, j 원소 제외)
<code>s[i:j:k]</code>	위치 i에서 위치 j까지 k 간격으로 띄운 시퀀스 조각 (i 원소 포함, j 원소 제외)
<code>len(s)</code>	s의 길이
<code>min(s)</code>	s에서 가장 작은 원소
<code>max(s)</code>	s에서 가장 큰 원소
<code>s.index(x)</code>	s에서 가장 앞에 나오는 원소 x의 인덱스 위치번호
<code>s.index(x, i)</code>	s의 위치 i 에서 시작하여 가장 앞에 나오는 원소 x의 인덱스 위치번호
<code>s.index(x, i, j)</code>	s의 위치 i 로부터 위치 j 전까지에서 가장 앞에 나오는 원소 x의 인덱스 위치번호
<code>s.count(x)</code>	s에서 원소 x의 빈도수
<code>s + t</code>	s와 t를 나란히 붙이기
<code>s * n</code>	s를 n번 반복하여 나란히 붙이기
<code>n * s</code>	s를 n번 반복하여 나란히 붙이기

# for 루프

## 문법

```
for <변수> in <시퀀스>:
    <블록>
```

## 의미

<변수>를 x라 하고, <시퀀스>를 s라 하면, 다음차례로 실행

- s[0]을 변수 x로 지정하고 <블록>을 실행한다.
- s[1]을 변수 x로 지정하고 <블록>을 실행한다.
- s[2]를 변수 x로 지정하고 <블록>을 실행한다.
- ...
- s[len(s)-1]을 변수 x로 지정하고 <블록>을 실행한다.  
(여기서 len(s)는 시퀀스 s의 길이를 리턴한다.)

<블록> 실행 중에 return 이나 break 문을 만나면 반복을 중단하고 루프를 빠져나간다.

- return : 값을 리턴하면서 함수 바깥으로 빠짐
- break : 포함된 블록을 감싸는 루프의 바깥으로 빠짐

# 리스트 정렬

# 리스트의 귀납 정의

- (1) 기초<sub>base</sub> :  $[]$ 은 리스트이다.
- (2) 귀납<sub>induction</sub> :  
 $x$ 가 임의의 원소이고  $xs$ 가 임의의 리스트이면,  $[x] + xs$ 도 리스트이다.  
여기서  $x$ 는 선두원소<sub>head</sub>,  $xs$ 는 후미리스트<sub>tail</sub>라고 한다.
- (3) 그 외에 다른 리스트는 없다.



# 선택정렬

리스트 `xs`를 정렬하려면,

(반복 조건) `xs != []`

(1) `xs`에서 가장 작은 원소를 찾아서 `smallest`로 지정하고,

(2) `xs`에서 `smallest`를 제거하고,

(3) **`xs`**를 재귀로 정렬하고,

(4) `smallest`와 정렬된 `xs`를 나란히 붙여서 리턴한다.

(종료 조건) `xs == []`

정렬할 원소가 없으므로 `[]`를 그대로 리턴한다.

- [\[Python 인터프리터\]](#)

- `min([3,5,4,2])`

- `xs.remove(x)` # `xs`에서 가장 앞에 나오는 원소 `x` 하나를 제거

# 선택정렬

리스트 `xs`를 정렬하려면,

(반복 조건) `xs != []`

- (1) `xs`에서 가장 작은 원소를 찾아서 `smallest`로 지정하고,
- (2) `xs`에서 `smallest`를 제거하고,
- (3) **`xs`**를 재귀로 정렬하고,
- (4) `smallest`와 정렬된 `xs`를 나란히 붙여서 리턴한다.

(종료 조건) `xs == []`

정렬할 원소가 없으므로 `[]`를 그대로 리턴한다.

```
1  def selectionSort(xs):
2      if xs != []:
3          smallest = min(xs)
4          xs.remove(smallest)
5          return [smallest] + selectionSort(xs)
6      else:
7          return []
```

```
1  def selectionSort(xs):
2      if xs != []:
3          smallest = min(xs)
4          xs.remove(smallest)
5          return [smallest] + selectionSort(xs)
6      else:
7          return []
```

### 함수 호출의 실행 추적

```
selectionSort([3,5,4,2])
⇒ [2] + selectionSort([3,5,4])
⇒ [2] + [3] + selectionSort([5,4])
⇒ [2] + [3] + [4] + selectionSort([5])
⇒ [2] + [3] + [4] + [5] + selectionSort([])
⇒ [2] + [3] + [4] + [5] + []
...
⇒ [2,3,4,5]
```

`xs.remove(x)`는 프로시저여서 아무것도 리턴하지 않음에 주의!

정확히는 None을 리턴

```
1  def selectionSort(xs):
2      if xs != []:
3          smallest = min(xs)
4          xs = xs.remove(smallest)
5          return [smallest] + selectionSort(xs)
6      else:
7          return []
```

```
1  def selectionSort(xs):
2      if xs != []:
3          smallest = min(xs)
4          return [smallest] + selectionSort(xs.remove(smallest))
5      else:
6          return []
```

## 재귀

```
1  def selectionSort(xs):
2      if xs != []:
3          smallest = min(xs)
4          xs.remove(smallest)
5          return [smallest] + selectionSort(xs)
6      else:
7          return []
```

## 꼬리재귀

```
1  def selectionSort(xs):
2      def loop(xs,ss):
3          if xs != []:
4              smallest = min(xs)
5              xs.remove(smallest)
6              return loop(xs, ss + [smallest])
7      else:
8          return ss
9      return loop(xs, [])
```

```
1  def selectionSort(xs):
2      def loop(xs,ss):
3          if xs != []:
4              smallest = min(xs)
5              xs.remove(smallest)
6              return loop(xs, ss + [smallest])
7      else:
8          return ss
9      return loop(xs, [])
```

## 함수 호출의 실행 추적

selectionSort([3,5,4,2])

⇒ loop([3,5,4,2], [])

⇒ loop([3,5,4], [2])

⇒ loop([5,4], [2,3])

⇒ loop([5], [2,3,4])

⇒ loop([], [2,3,4,5])

⇒ [2,3,4,5]

```
1  def selectionSort(xs):
2      def loop(xs,ss):
3          if xs != []:
4              smallest = min(xs)
5              xs.remove(smallest)
6              return loop(xs, ss + [smallest])
7      else:
8          return ss
9      return loop(xs, [])
```

```
1  def selectionSort(xs):
2      def loop(xs,ss):
3          if xs != []:
4              smallest = min(xs)
5              xs.remove(smallest)
6              ss.append(smallest)
7              return loop(xs, ss)
8      else:
9          return ss
10     return loop(xs, [])
```

- [[Python 인터프리터](#)]

- `xs.append(x)` # `xs`의 맨 뒤에 원소 `x`를 추가, procedure 임에 주의!

# 삽입정렬

리스트 `xs`를 정렬하려면,

(반복 조건) `xs != []`

(1) `xs`의 후미리스트인 `xs[1:]`을 재귀로 정렬하고,

(2) `xs`의 선두원소인 `xs[0]`을 정렬한 리스트의 적절한 위치에 끼워서 리턴한다.

(종료 조건) `xs == []`

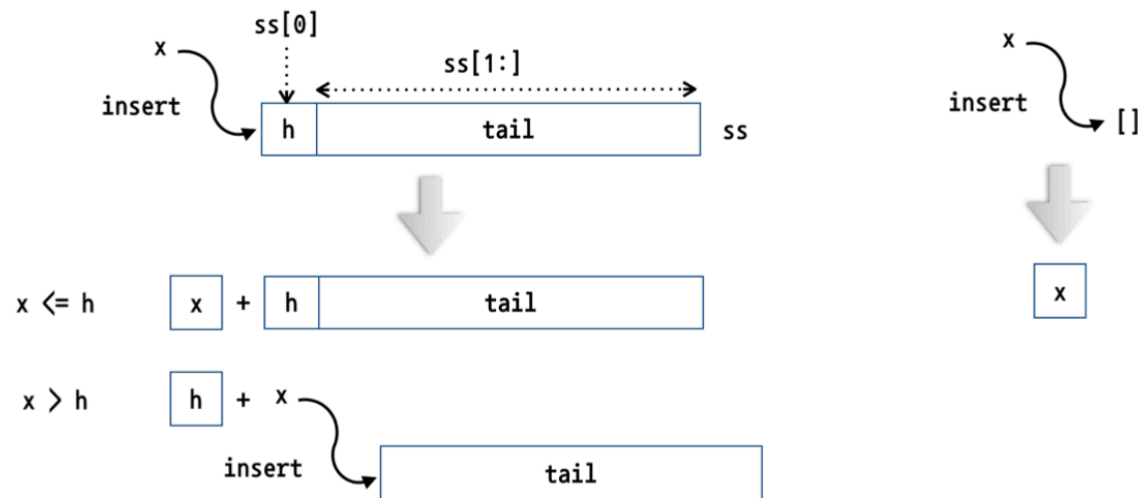
정렬할 원소가 없으므로 `[]`를 그대로 리턴한다.

```
1 def insertionSort(xs):
2     if xs != []:
3         return insert(xs[0], insertionSort(xs[1:]))
4     else:
5         return []
```



# 삽입정렬

<code>insert(x,ss)</code>	재귀함수 설계 전략	함수 호출 (예시)	계산 결과 (예시)
귀납 <code>ss != []</code>	ss의 선두원소보다 작으면 앞에 붙인다.	<code>insert(1,[2,4,5])</code>	<code>[1] + [2,4,5]</code>
	ss의 선두원소보다 크면 ss의 후미리스트에 재귀로 끼워 넣는다.	<code>insert(3,[2,4,5])</code>	<code>[2] + insert(3,[4,5])</code>
기초 <code>ss == []</code>	리스트로 만든다.	<code>insert(1,[])</code>	<code>[1]</code>



# 삽입정렬

<code>insert(x,ss)</code>	재귀함수 설계 전략	함수 호출 (예시)	계산 결과 (예시)
귀납 <code>ss != []</code>	<code>ss</code> 의 선두원소보다 작으면 앞에 붙인다.	<code>insert(1,[2,4,5])</code>	<code>[1] + [2,4,5]</code>
	<code>ss</code> 의 선두원소보다 크면 <code>ss</code> 의 후미리스트에 재귀로 끼워 넣는다.	<code>insert(3,[2,4,5])</code>	<code>[2] + insert(3,[4,5])</code>
기초 <code>ss == []</code>	리스트로 만든다.	<code>insert(1,[])</code>	<code>[1]</code>

원소 `x`를 정렬된 리스트 `ss`의 제 위치에 끼워 넣으려면,

(반복 조건) `ss != []`

`ss`를 선두원소 `ss[0]`와 후미리스트 `ss[1:]`로 나눈다.

(1) `x <= ss[0]` 이면, `x`를 `ss`의 앞에 붙여서 리턴한다.

(2) `x > ss[0]` 이면, 재귀로 `x`를 `ss[1:]`의 제 위치에 끼워 넣고

그 앞에 `ss[0]`를 붙여서 리턴한다.

(종료 조건) `ss == []`

`[x]`를 리턴한다.

# 삽입정렬

원소  $x$ 를 정렬된 리스트  $ss$ 의 제 위치에 끼워 넣으려면,

(반복 조건)  $ss \neq []$

$ss$ 를 선두원소  $ss[0]$ 와 후미리스트  $ss[1:]$ 로 나눈다.

(1)  $x \leq ss[0]$  이면,  $x$ 를  $ss$ 의 앞에 붙여서 리턴한다.

(2)  $x > ss[0]$  이면, 재귀로  $x$ 를  $ss[1:]$ 의 제 위치에 끼워 넣고  
그 앞에  $ss[0]$ 를 붙여서 리턴한다.

(종료 조건)  $ss = []$

$[x]$ 를 리턴한다.

```
1  def insert(x,ss):
2      if ss != []:
3          if x <= ss[0]:
4              return [x] + ss
5          else:
6              return [ss[0]] + insert(x, ss[1:])
7      else:
8          return [x]
```

# 합병정렬

리스트  $xs$ 를 합병정렬하려면,

(반복 조건)  $\text{len}(xs) > 1$

$xs$ 를 반으로 나누어, 각각 재귀로 합병정렬을 완료하고,

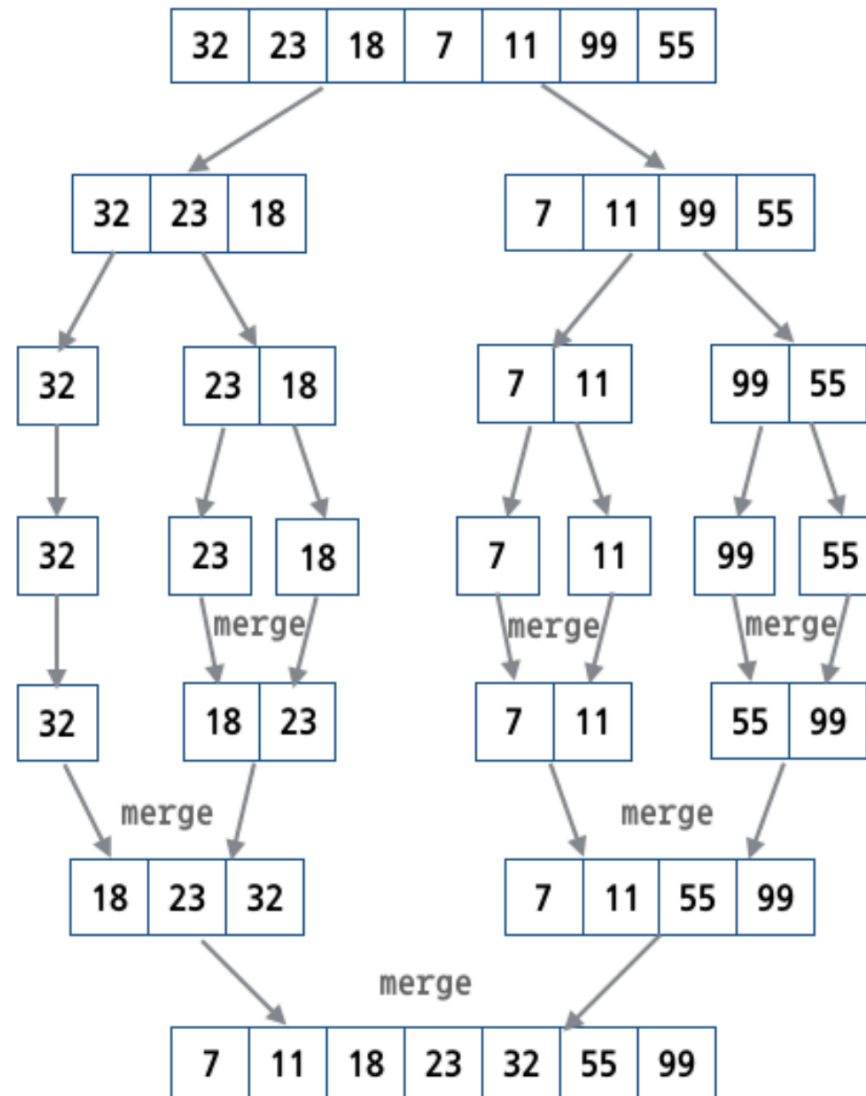
두 정렬된 리스트를 앞에서부터 차례로 훑어가며

가장 작은 원소를 먼저 선택하는 방식으로 하나로 합병(merge)하여 리턴한다.

(종료 조건)  $\text{len}(xs) \leq 1$

정렬할 필요가 없으므로 그대로 리턴한다.

# 합병정렬

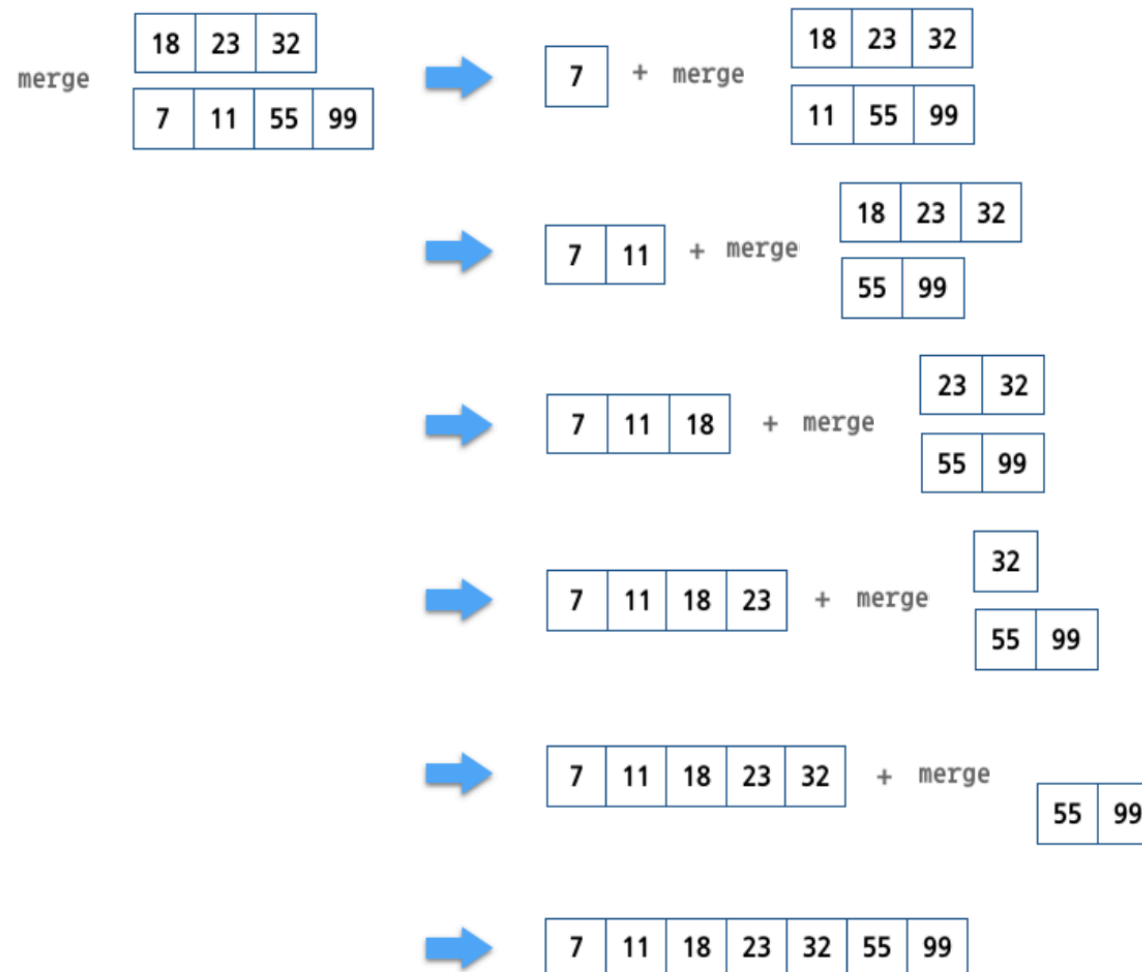


# 합병정렬

```
1 def mergeSort(xs):
2     if len(xs) > 1:
3         mid = len(xs) // 2
4         return merge(mergeSort(xs[:mid]), mergeSort(xs[mid:]))
5     else:
6         return xs
```

```
1 def merge(left, right):
2     if not (left == [] or right == []):
3         if left[0] <= right[0]:
4             return [left[0]] + merge(left[1:], right)
5         else:
6             return [right[0]] + merge(left, right[1:])
7     else:
8         return left + right
```

# 합병정렬



# 퀵정렬

리스트 `xs`를 정렬하려면,

(반복 조건) `len(xs) > 1`

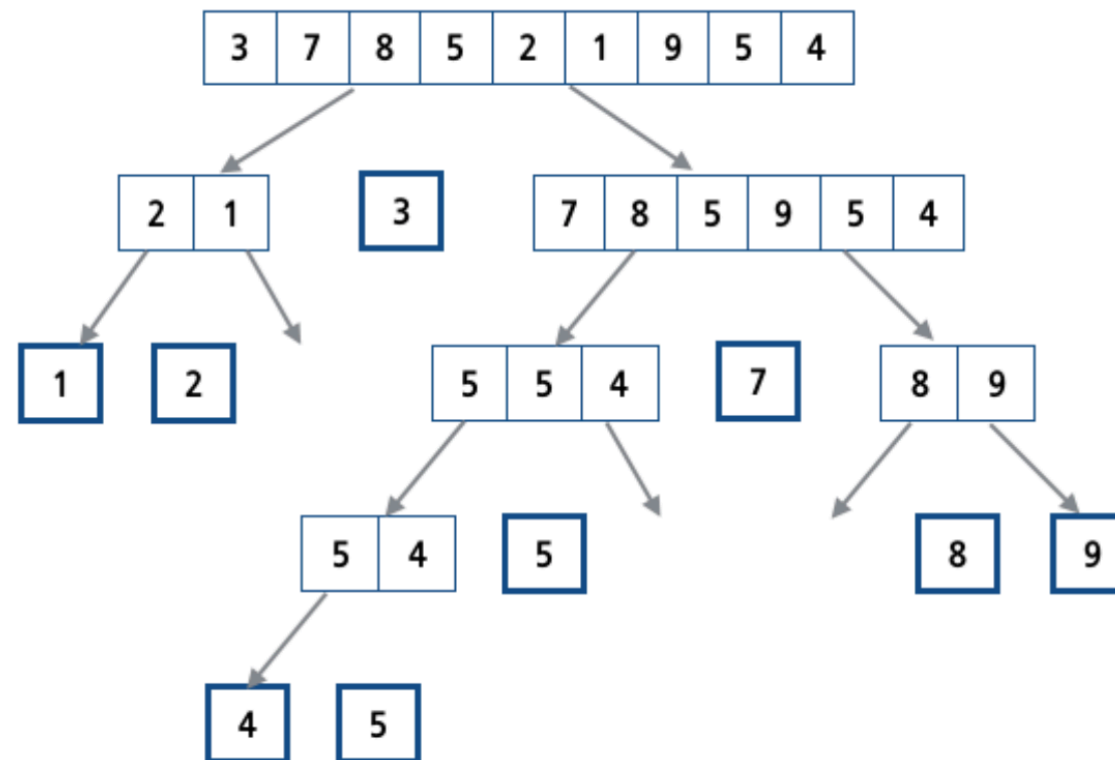
- (1) 기준으로 사용할 피벗 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다.
- (2) `pivot` 원소를 기준으로 작은 원소는 왼쪽 리스트 `ls`로, 큰 원소는 오른쪽 리스트 `rs`로 옮긴다.
- (3) 왼쪽 리스트 `ls`와 오른쪽 리스트 `rs`를 각각 재귀로 정렬하고, `ls`와 `pivot`과 `rs`를 나란히 붙여서 리턴한다.

(종료 조건) `len(xs) <= 1`

정렬할 필요가 없으므로 그대로 리턴한다.



# 퀵정렬



# 퀵정렬

```
1 def quicksort(xs):  
2     if len(xs) > 1:  
3         pivot = xs[0]  
4         left, right = partition(pivot, xs[1:])  
5         return quicksort(left) + [pivot] + quicksort(right)  
6     else:  
7         return xs
```

```
1 def partition(pivot, xs):  
2     if xs != []:  
3         left, right = partition(pivot, xs[1:])  
4         if xs[0] <= pivot:  
5             left.append(xs[0])  
6         else:  
7             right.append(xs[0])  
8         return left, right  
9     else:  
10        return [], []
```