>>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음

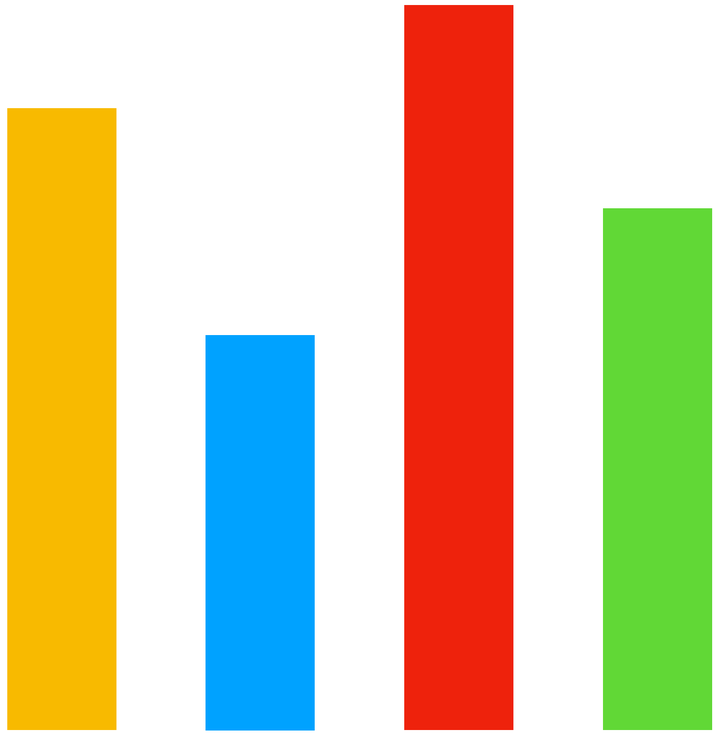
생능출판

CHAPTER 5

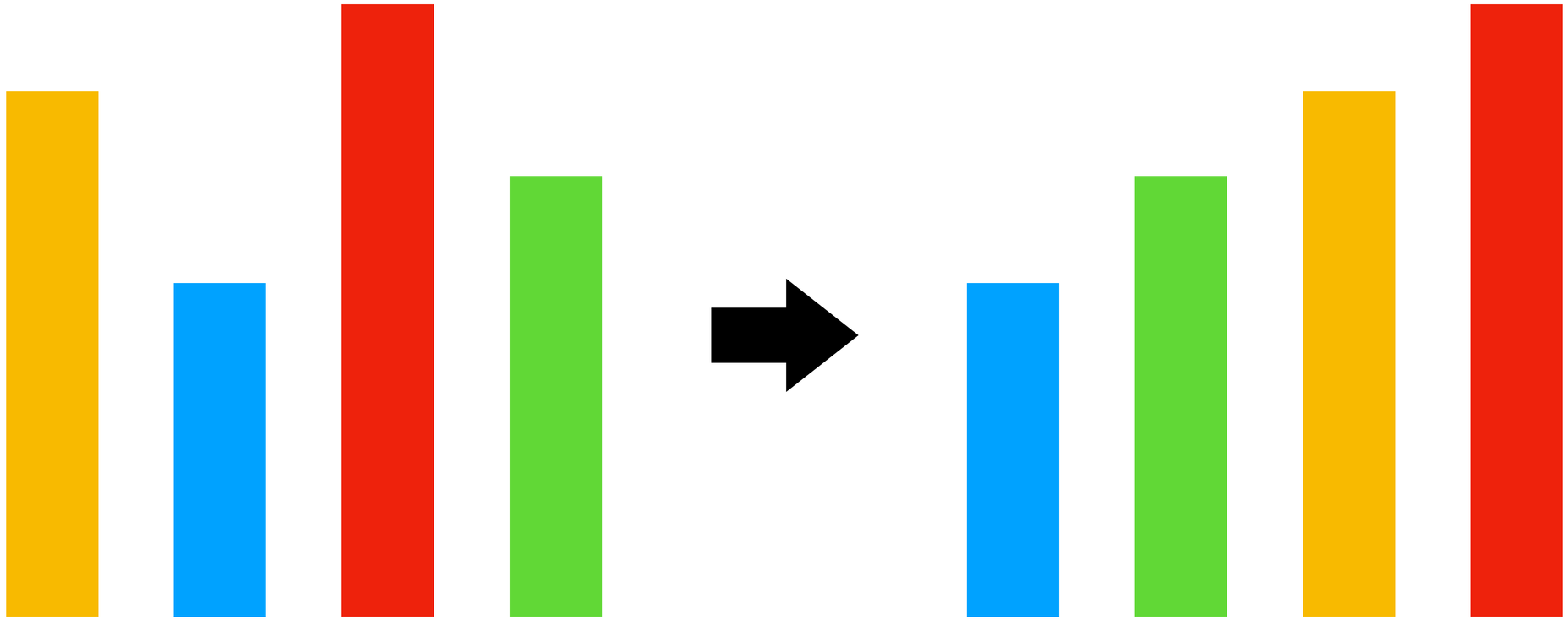
재귀와 반복 : 정렬

정렬

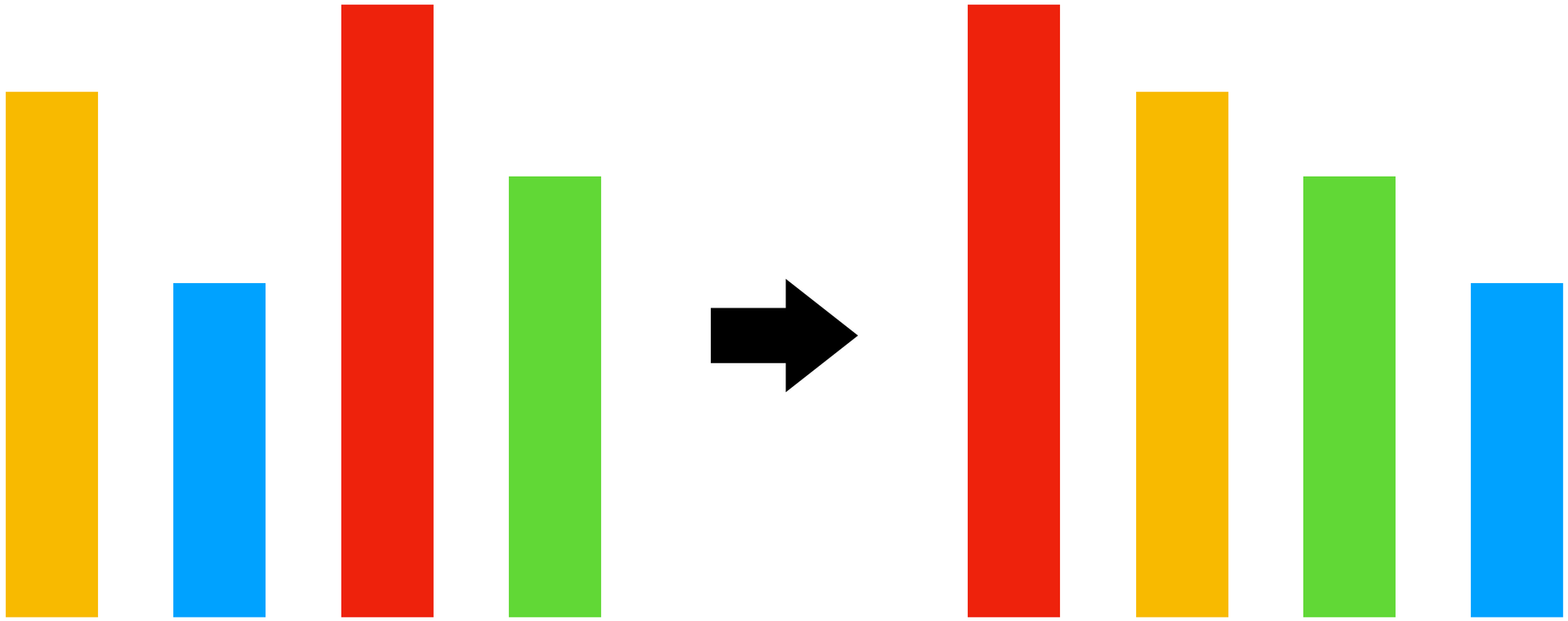
Sorting



정렬
Sorting



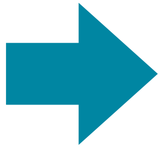
정렬
Sorting



리스트

List

```
[3, 5, 4, 2]
["컴퓨터과학", "대학교", "소프트웨어", "인공지능"]

ns.sort()
```



CHAPTER 5

재귀와 반복 : 정렬

✔ 5.1 시퀀스

5.2 리스트 정렬

시퀀스

Sequence

시퀀스 타입 Sequence Type	**리스트** List	`[4, 6, 9, 11]`	
	튜플 Tuple	`('컴퓨터과학', 1, '짱')`	
	정수범위 Range	`range(3,9)`	

시퀀스

Sequence

시퀀스 타입 Sequence Type	**리스트** List	`[4, 6, 9, 11]`	
	튜플 Tuple	`('컴퓨터과학', 1, '짱')`	
	정수범위 Range	`range(3,9)`	
텍스트 시퀀스 타입 Text Sequence Type	**문자열** String	`"컴퓨터과학"`	

시퀀스

Sequence

시퀀스 타입 Sequence Type	**리스트** List	`[4, 6, 9, 11]`	**수정 가능** Mutable
	튜플 Tuple	`('컴퓨터과학', 1, '짱')`	**수정 불가능** Immutable
	정수범위 Range	`range(3,9)`	
텍스트 시퀀스 타입 Text Sequence Type	**문자열** String	`"컴퓨터과학"`	

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

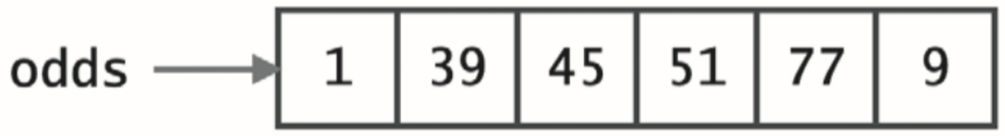
```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```



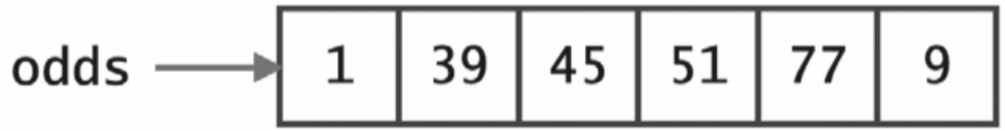
```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```



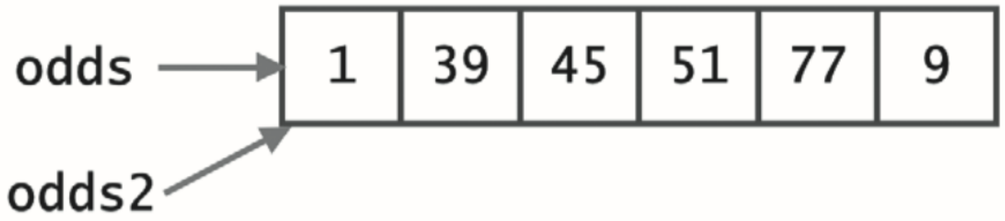
```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

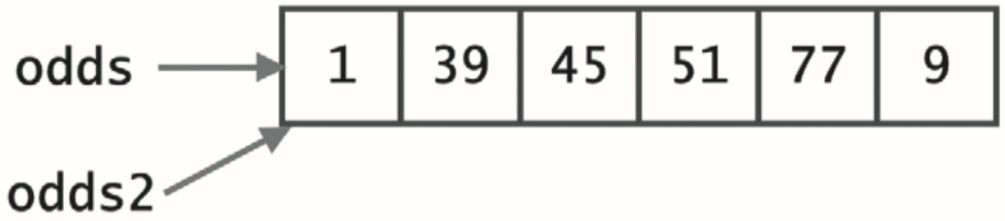
```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

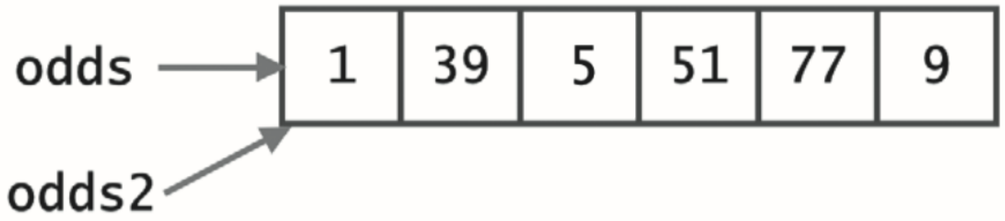
```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

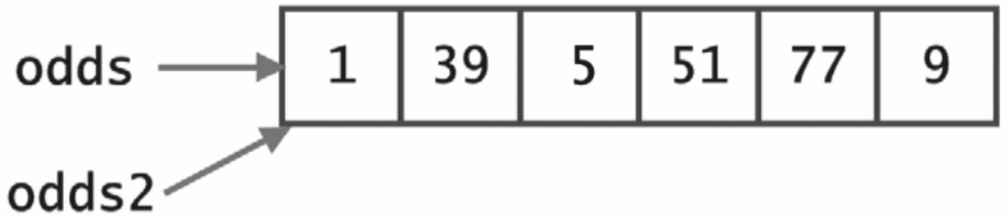
```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

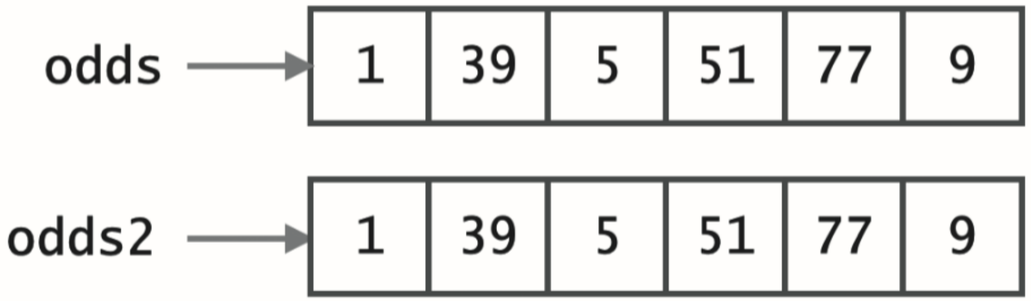
```
>>> odds[1:3] = [11+22, 22+33]
```

```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

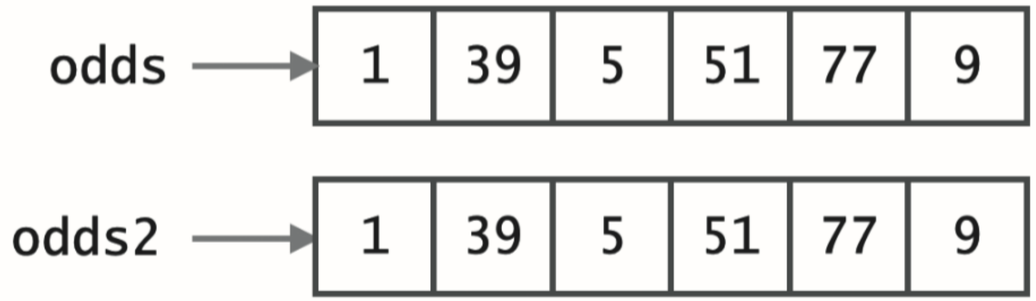
```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

```
>>> odds2 = odds[:]
```

```
>>> odds2[4] = 7
```



```
>>> odds = [1, 1+2, 2+3, 3+4, 4+5]
```

```
>>> odds[3] = 33 + 44
```

```
>>> odds[1:3] = [11+22, 22+33]
```

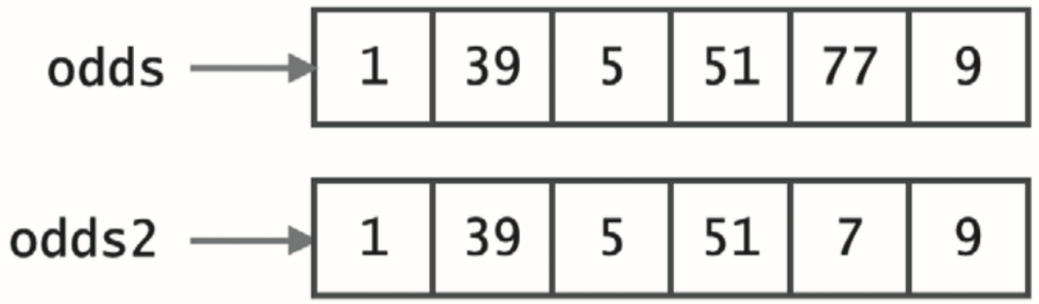
```
>>> odds[1:3] = [11+28,22+23,23+28]
```

```
>>> odds2 = odds
```

```
>>> odds2[2] = 5
```

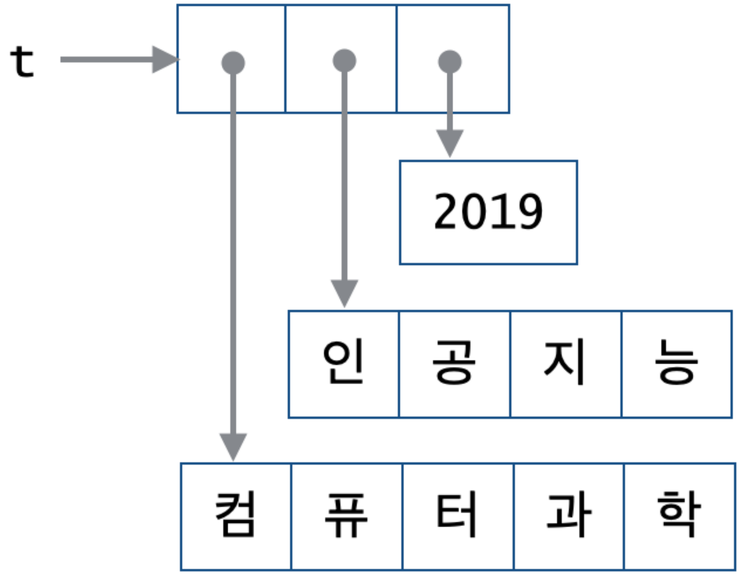
```
>>> odds2 = odds[:]
```

```
>>> odds2[4] = 7
```



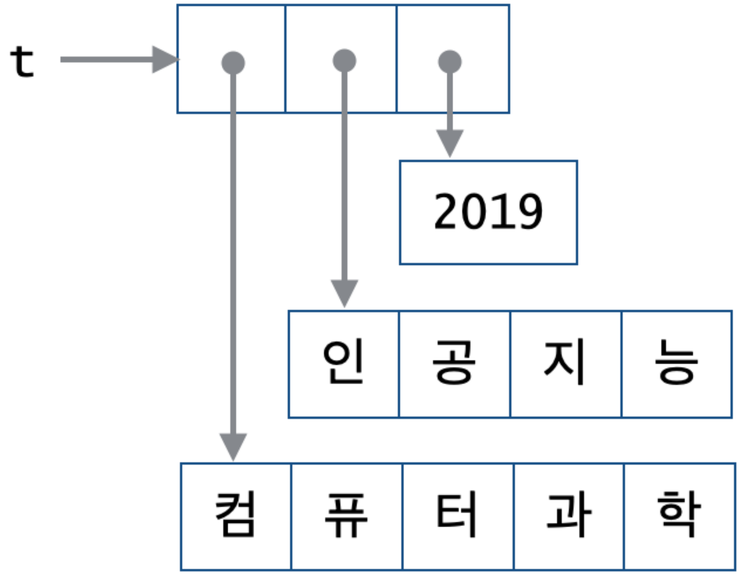
```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```



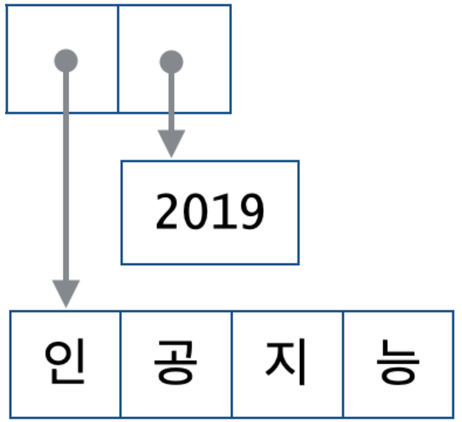
```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

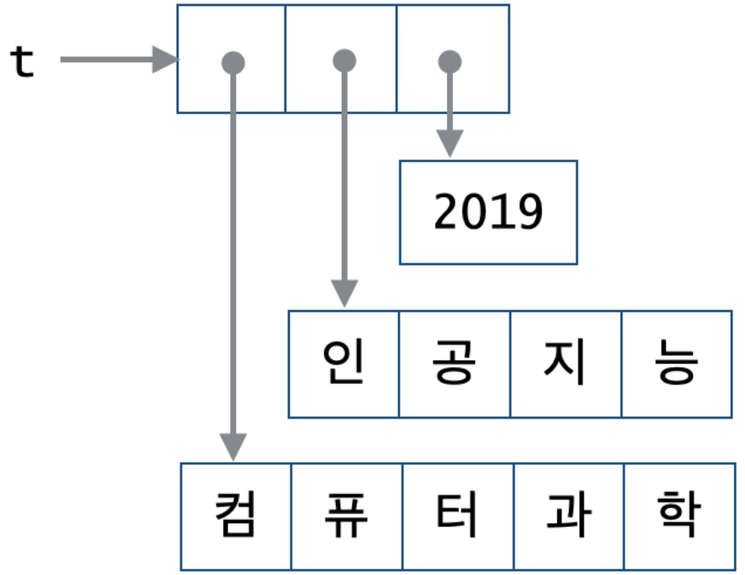
```
>>> t[1:]
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

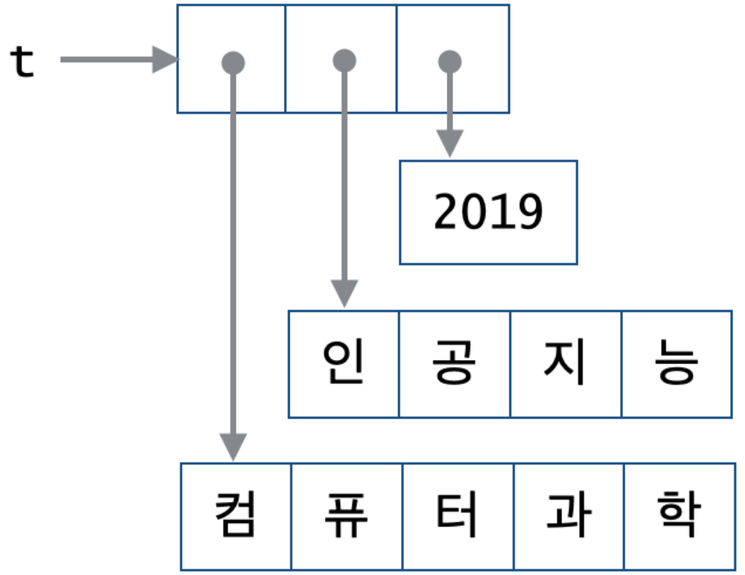




```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

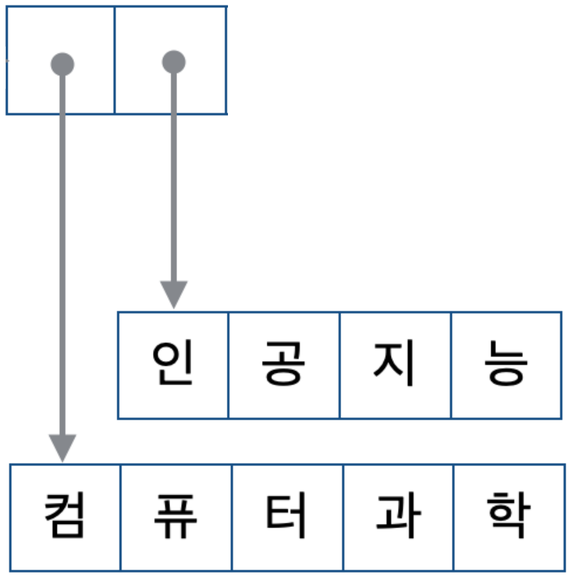
```
>>> t[:2]
```

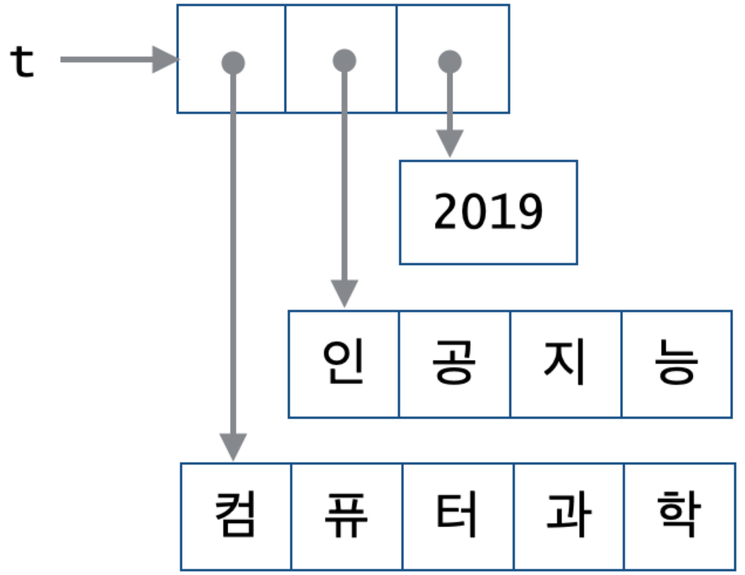


```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```



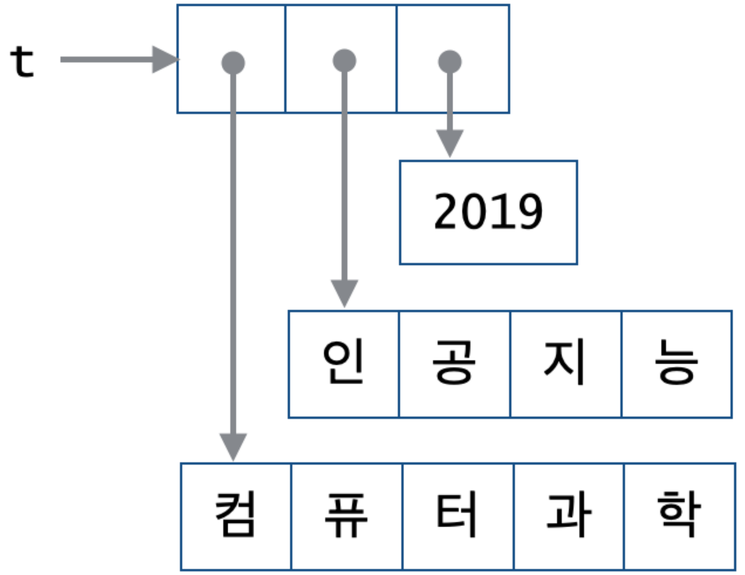


```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```



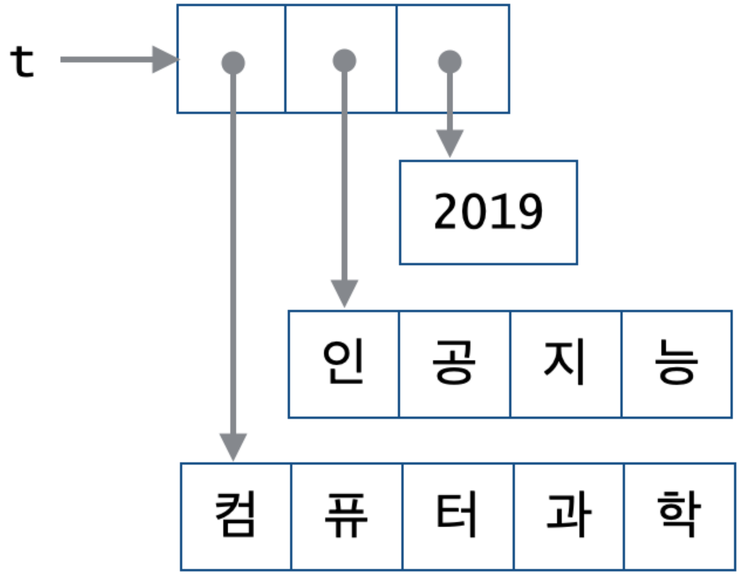
```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

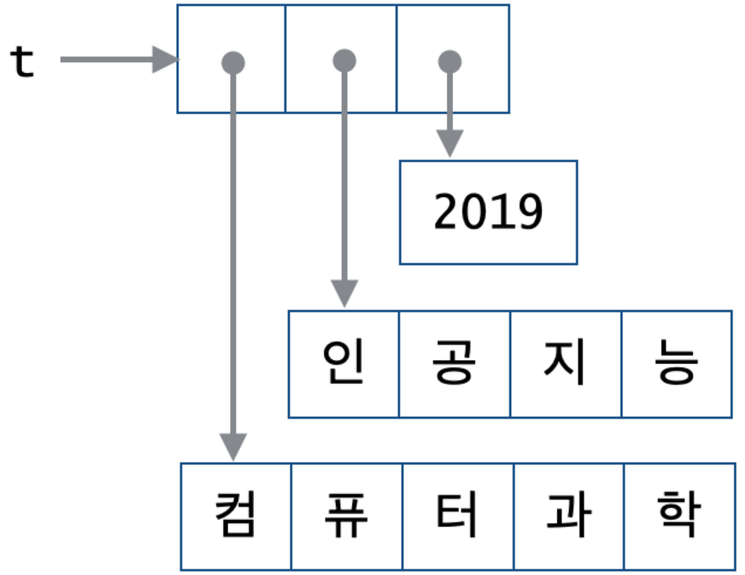
```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

a	l	o	n	e



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

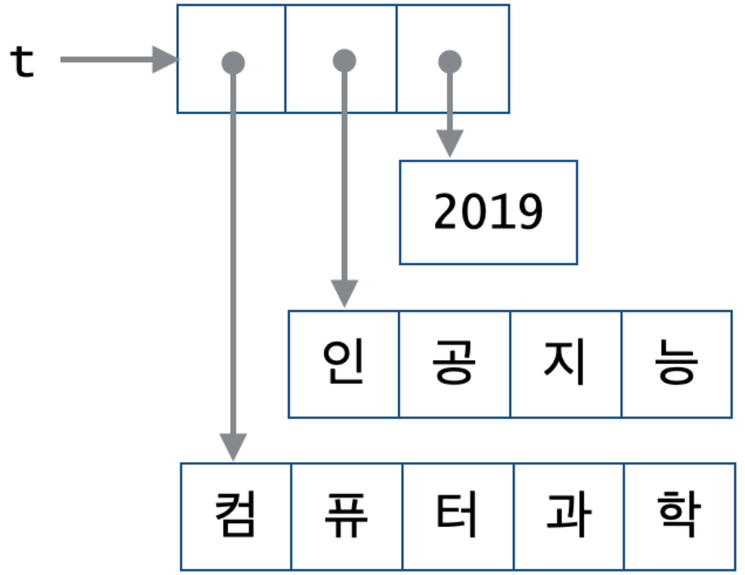
```
>>> t[1:]
```

```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

```
>>> ("alone",)
```



```
>>> t = ('컴퓨터과학', '인공지능', 2019)
```

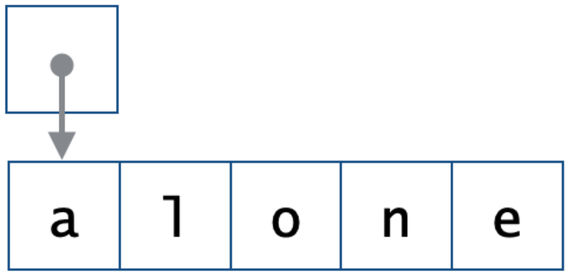
```
>>> t[1:]
```

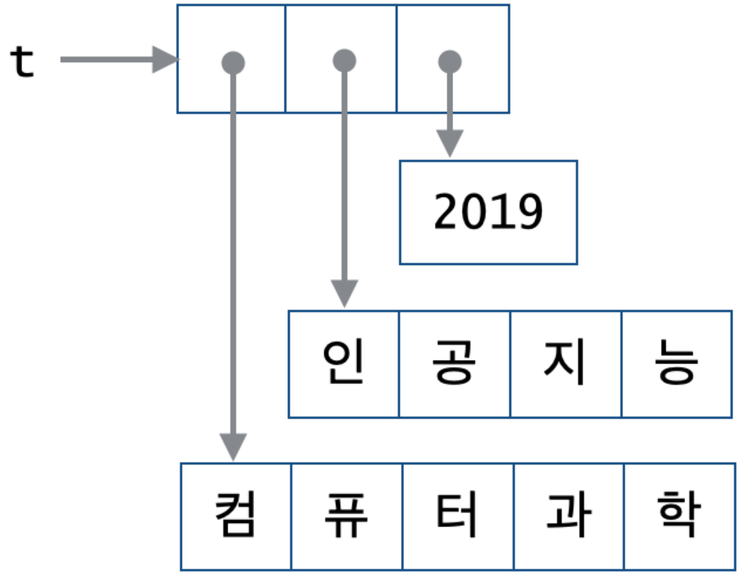
```
>>> t[:2]
```

```
>>> t[2] = '꽝'
```

```
>>> ("alone")
```

```
>>> ("alone",)
```





```
>>> s = "컴퓨터과학"
```

```
>>> s = "컴퓨터과학"
```

```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```

```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```

```
>>> s = "컴퓨터과학"
```

```
>>> s[3] = "공"
```

```
>>> s = s + "도"
```

- range(n)은 정수 0부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(5)
```

- range(m,n)은 정수 m부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(3,10)
```

- range(m,n,k)는 정수 m부터 n-1까지를 간격 k로 나열한 정수 범위를 나타낸다.

```
range(3,11,2)
```

```
range(10,3,-1)
```

- range(n)은 정수 0부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(5)
```

- range(m,n)은 정수 m부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(3,10)
```

- range(m,n,k)는 정수 m부터 n-1까지를 간격 k로 나열한 정수 범위를 나타낸다.

```
range(3,11,2)
```

```
range(10,3,-1)
```

- range(n)은 정수 0부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(5)
```

- range(m,n)은 정수 m부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(3,10)
```

- range(m,n,k)는 정수 m부터 n-1까지를 간격 k로 나열한 정수 범위를 나타낸다.

```
range(3,11,2)
```

```
range(10,3,-1)
```

- range(n)은 정수 0부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

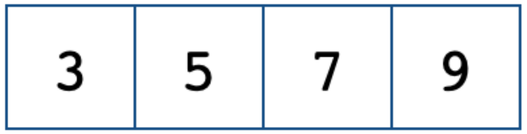
```
range(5)
```

- range(m,n)은 정수 m부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

```
range(3,10)
```

- range(m,n,k)는 정수 m부터 n-1까지를 간격 k로 나열한 정수 범위를 나타낸다.

```
range(3,11,2)
```




```
range(10,3,-1)
```

- range(n)은 정수 0부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

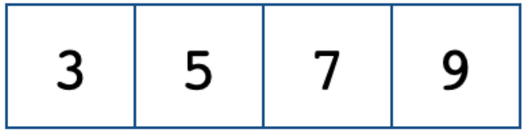
range(5)

- range(m,n)은 정수 m부터 n-1까지를 간격 1로 나열한 정수 범위를 나타낸다.

range(3,10)

- range(m,n,k)는 정수 m부터 n-1까지를 간격 k로 나열한 정수 범위를 나타낸다.

range(3,11,2)



range(10,3,-1)

시퀀스 연산

s, t : 시퀀스
x : 원소
i, j, k : 인덱스
n : 자연수

연산	의미
`x in s`	x가 s에 있으면 True, 없으면 False
`x not in s`	x가 s에 없으면 True, 있으면 False
`s[i]`	s에서 i에 있는 원소
`s[i:j]`	i 에서 j 까지 시퀀스 조각 (i 포함, j 제외)
`s[i:j:k]`	i 에서 j 까지 k간격으로 띄운 시퀀스 조각 (i 포함, j 제외)
`len(s)`	s의 길이
`min(s)`	s에서 가장 작은 원소
`max(s)`	s에서 가장 큰 원소
`s.index(x)`	s에서 가장 앞에 나오는 x의 인덱스
`s.index(x,i)`	s의 i 에서 시작하여 가장 앞에 나오는 x의 인덱스
`s.index(x,i,j)`	s의 i에서 j까지 범위에서 가장 앞에 나오는 x의 인덱스 (i 포함, j 제외)
`s.count(x)`	s에서 x의 빈도수
`s + t`	s와 t 나란히 붙이기
`s * n`	s를 n번 반복하여 나란히 붙이기
`n * s`	s를 n번 반복하여 나란히 붙이기

for 루프

for-loop

```
for 〈변수〉 in 〈시퀀스〉:
    〈블록〉
```

for 루프

for-loop

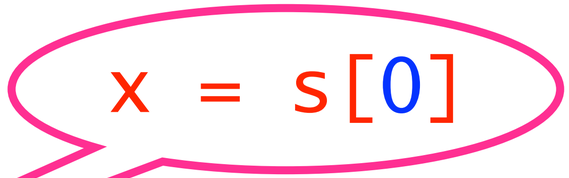
```
for x in s:
```

〈블록〉

for 루프

for-loop

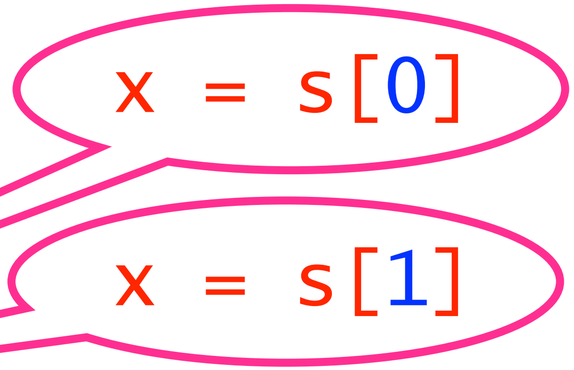
```
for x in s:
    〈블록〉
```



for 루프

for-loop

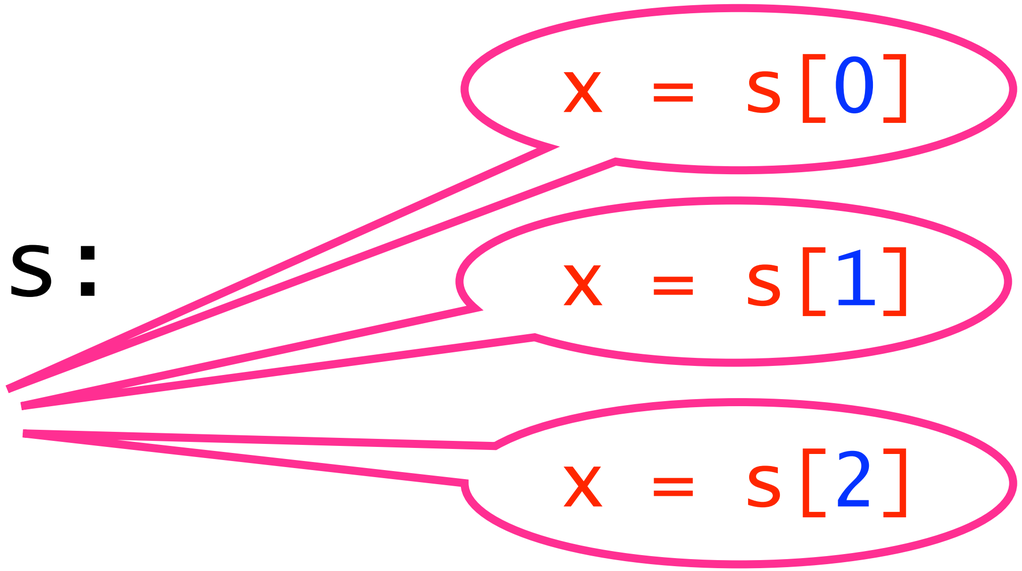
```
for x in s:
    〈블록〉
```



for 루프

for-loop

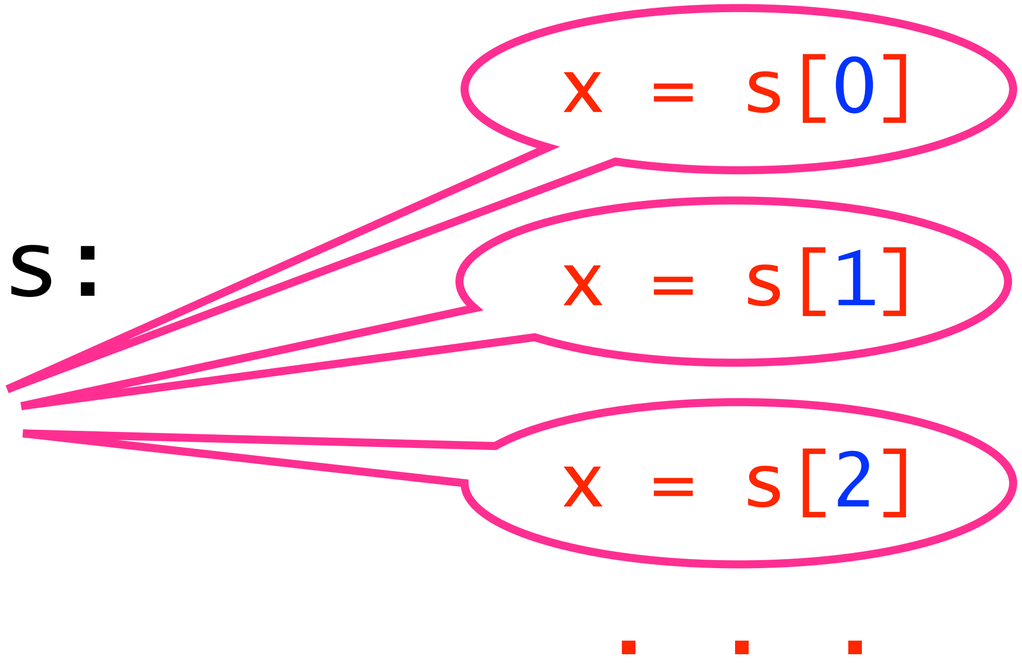
```
for x in s:
    〈블록〉
```



for 루프

for-loop

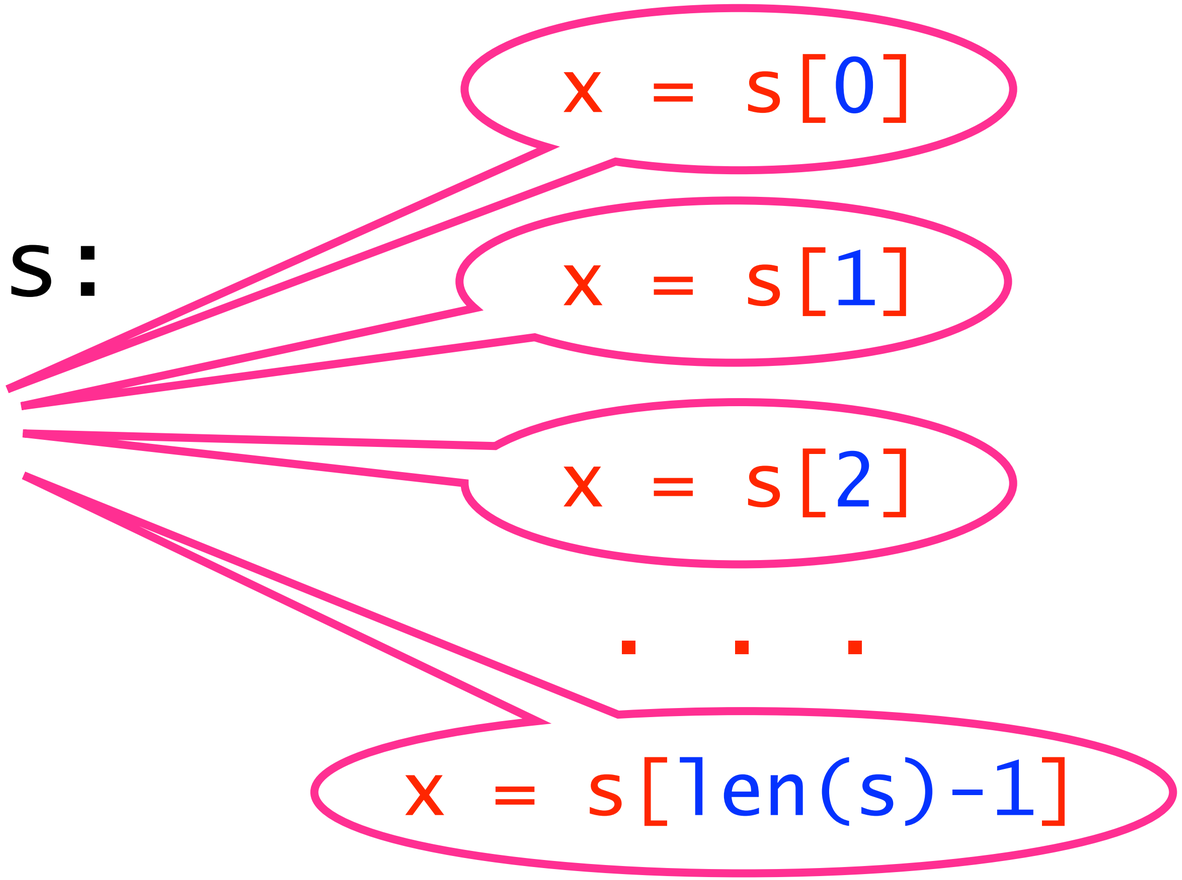
```
for x in s:
    〈블록〉
```



for 루프

for-loop

```
for x in s:
    〈블록〉
```



실습 5.1 for 루프 작성 연습

4장에서 while 문으로 작성했던 함수를 왼쪽에 나열해놓았다. 이 함수를 for 루프를 사용하여 오른쪽 뼈대코드의 빈칸을 채워서 재작성해 보자. 두 함수는 실행 의미가 동일해야 한다.

카운트다운 타이머

code : 5-1.py

```
1  from time import sleep
2  def countdown(n):
3      while n > 0:
4          print(n)
5          sleep(1)
6          n = n - 1
7      print("Launch!")
```

code : 5-2.py

```
1  from time import sleep
2  def countdown(n):
3      for i in range(n, ,  ):
4          print(i)
5          sleep(1)
6      print("Launch!")
7
```

자연수 수열의 합 (순진무구 알고리즘)

code : 5-3.py

```
1  def sigma(n):
2      sum = 0
3      while n > 0:
4          n, sum = n-1, n+sum
5      return sum
```

code : 5-4.py

```
1  def sigma(n):
2      sum = 0
3      for           :
4
5      return sum
```

구간 수열의 합

code : 5-5.py

```
1  def sumrange(m,n):
2      sum = 0
3      while m <= n:
4          sum = sum + m
5          m = m + 1
6      return sum
```

code : 5-6.py

```
1  def sumrange(m,n):
2      sum = 0
3      for           :
4
5      return sum
6
```

팩토리얼

code : 5-7.py

```
1  def fac(n):
2      ans = 1
3      while n > 0:
4          n, ans = n-1, ans*n
5      return ans
```

code : 5-8.py

```
1  def fac(n):
2      ans = 1
3      for           :
4
5      return ans
```

거듭제곱 (순진무구 알고리즘)

code : 5-9.py

```
1  def power(b,n):
2      prod = 1
3      while n > 0:
4          prod = b * prod
5          n = n - 1
6      return prod
```

code : 5-10.py

```
1  def power(b,n):
2      prod = 1
3      for           :
4
5      return prod
6
```

>>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음

생능출판

CHAPTER 5

재귀와 반복 : 정렬

CHAPTER 5

재귀와 반복 : 정렬

5.1 시퀀스

✔ 5.2 리스트 정렬

순서를 매길 수 있는 원소로 구성된 리스트

정렬
Sorting

순서대로 정렬한 리스트

리스트 vs. 배열

	인덱스로 직접 접근	길이	원소의 타입
리스트 List	불가능	가변	달라도 됨
배열 Array	가능	고정	같아야 함

파이썬의 리스트

	인덱스로 직접 접근	길이	원소의 타입
리스트 List	불가능	가변	달라도 됨
배열 Array	가능	고정	같아야 함

리스트
List
구조 귀납歸納,인덕
Structural Induction

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.
(3)	그 외에 다른 리스트는 없다.	

(1)	**기초** **Basis**	빈 리스트 `[]`은 **리스트** 이다.
(2)	**귀납** **Induction**	`x`가 임의의 **원소**이고 `xs`가 임의의 **리스트**이면, `x :: xs` 도 **리스트**이다. 여기서, `::`는 리스트 생성 연산자로 **cons**tructor 라고 한다. `x`는 **선두원소(head)**, `xs`은 **후미리스트(tail)**라고 한다.

`[]`

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `x :: xs` 도 **리스트**이다. 여기서, `::`는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
2 :: []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
4 :: 2 :: []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
5 :: 4 :: 2 :: []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

[2]

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

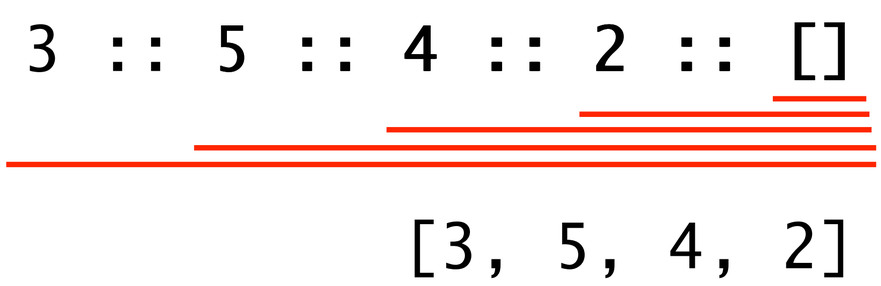
```
[4, 2]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

```
[5, 4, 2]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.



(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []
```

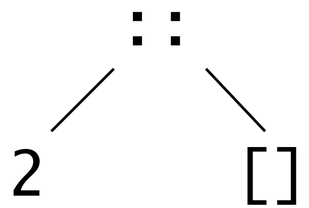
```
[3, 5, 4, 2]
```

```
[]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

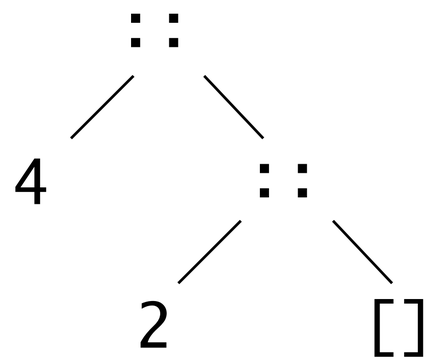
```
3 :: 5 :: 4 :: 2 :: []
```

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



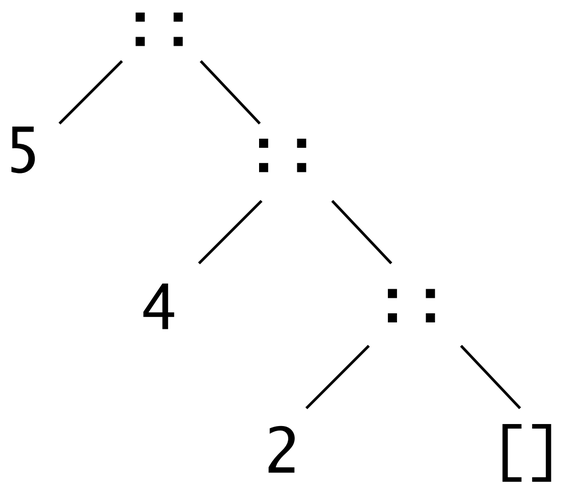
(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []

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



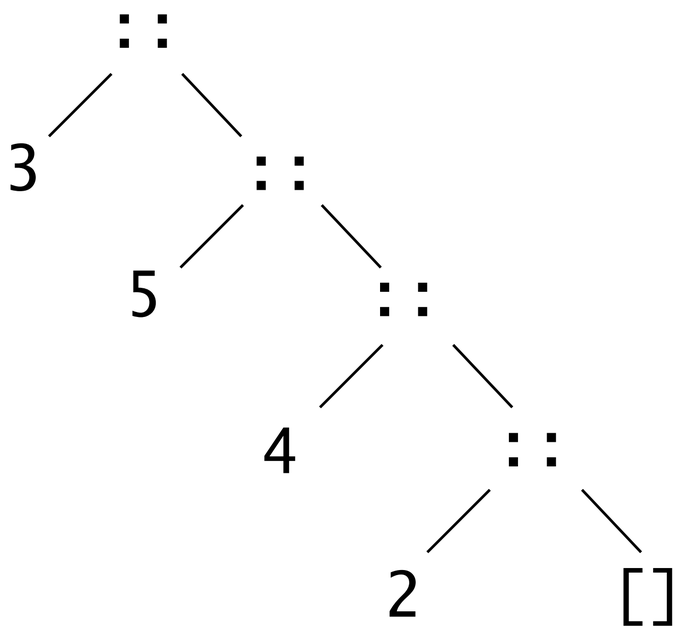
(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []

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



(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, x :: xs 도 **리스트**이다. 여기서, ::는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
3 :: 5 :: 4 :: 2 :: []

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



(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `x :: xs` 도 **리스트**이다. 여기서, `::`는 리스트 생성 연산자로 **cons**tructor 라고 한다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, [x] + xs 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

[]

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

`[2] + []`

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[2] + []
```

```
[2]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[4] + [2] + []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[4] + [2] + []
```

```
[4, 2]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, [x] + xs 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[5] + [4] + [2] + []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[5] + [4] + [2] + []
```

```
[5, 4, 2]
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, [x] + xs 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[3] + [5] + [4] + [2] + []
```

(1)	**기초** **Basis**	빈 리스트 []은 **리스트** 이다.
(2)	**귀납** **Induction**	x가 임의의 **원소**이고 xs가 임의의 **리스트**이면, `[x] + xs` 도 **리스트**이다. x는 **선두원소(head)**, xs은 **후미리스트(tail)**라고 한다.

```
[3] + [5] + [4] + [2] + []
```

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

선택정렬

Selection Sort

선택정렬 알고리즘
Selection Sort

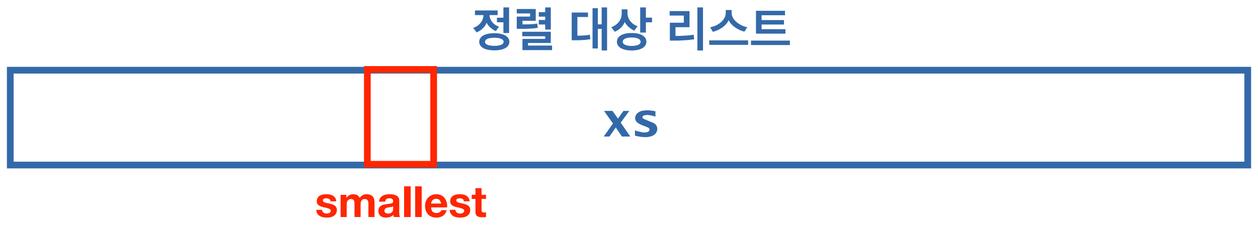
리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

리스트 xs를 정렬 하려면		
반복 조건	xs != []	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

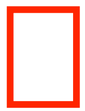
정렬 대상 리스트

리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

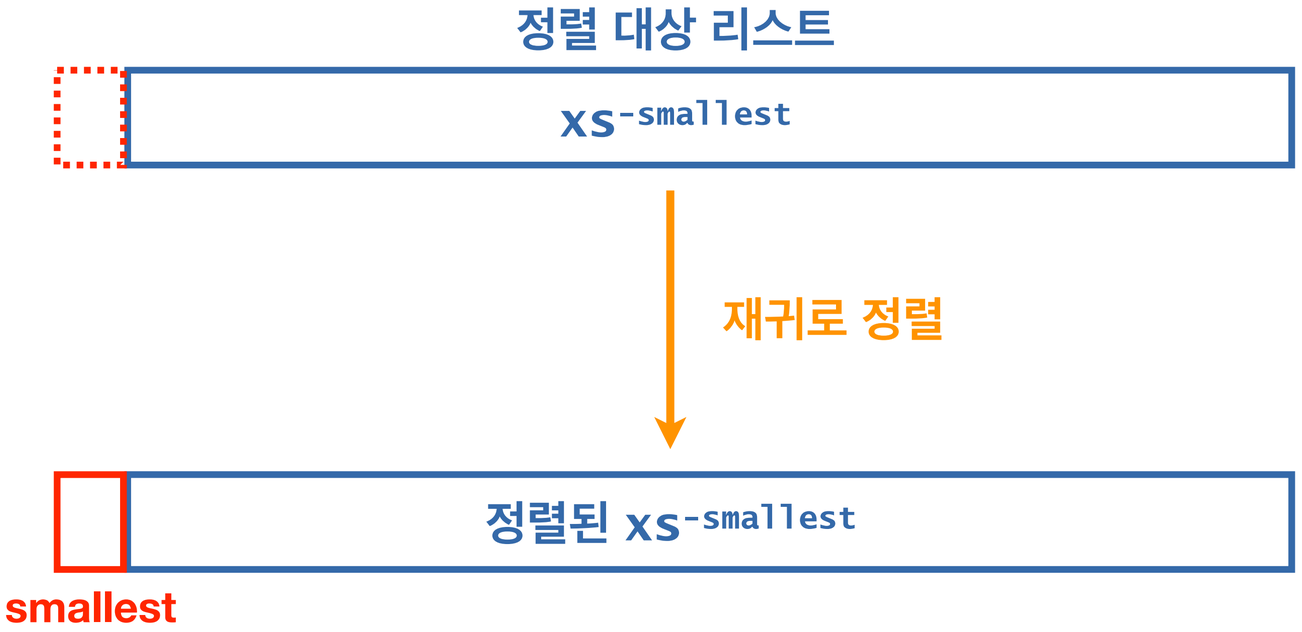


리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

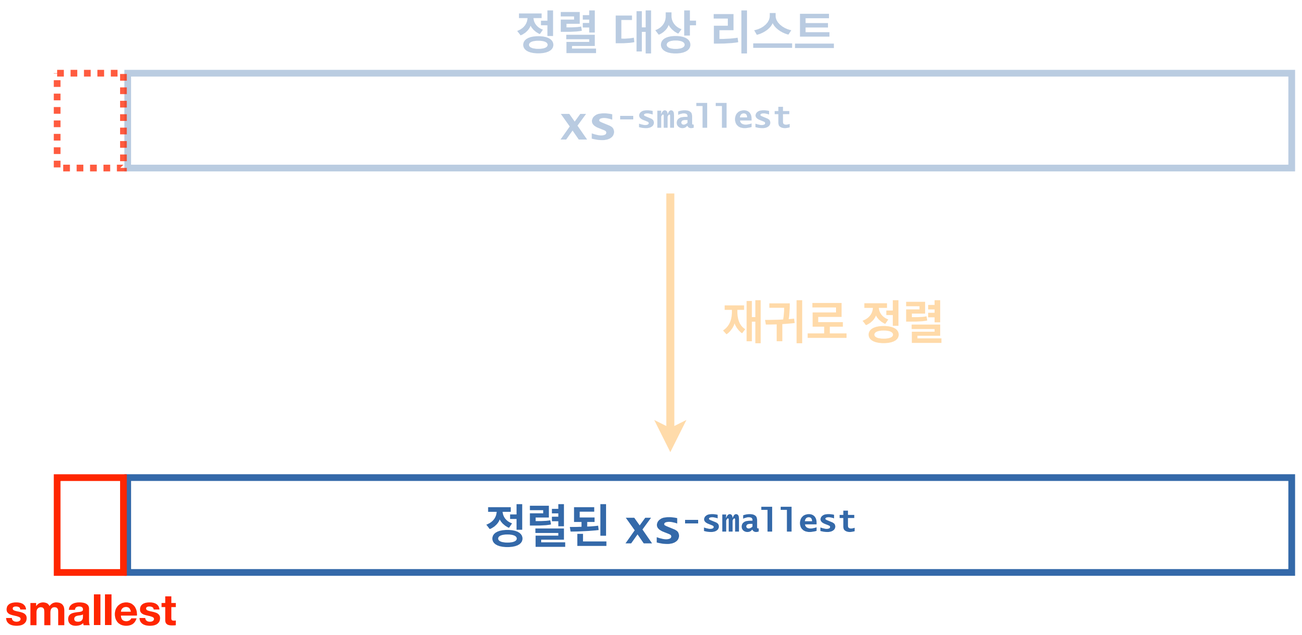
리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	xs != []	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

정렬된 xs

리스트 xs를 정렬 하려면		
반복 조건	`xs != []`	• xs에서 가장 작은 원소를 찾아서 smallest로 지정하고, • xs에서 smallest를 제거하고, • xs를 재귀로 정렬하고, • smallest가 선두원소, 정렬된 xs가 후미리스트인 리스트를 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

code : 5-11.py

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

리스트 메소드

연산	의미	비고
`xs.remove(x)`	`xs`에서 가장 앞에 나오는 원소 `x`을 제거한다.	`x`이 `xs`에 없으면 `ValueError`가 발생한다.

code : 5-11.py

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

```
selection_sort([3,5,4,2])
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
=> [2] + [3] + selection_sort([5,4])
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
=> [2] + [3] + selection_sort([5,4])
=> [2] + [3] + [4] + selection_sort([5])
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
=> [2] + [3] + selection_sort([5,4])
=> [2] + [3] + [4] + selection_sort([5])
=> [2] + [3] + [4] + [5] + selection_sort([])
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
=> [2] + [3] + selection_sort([5,4])
=> [2] + [3] + [4] + selection_sort([5])
=> [2] + [3] + [4] + [5] + selection_sort([])
=> [2] + [3] + [4] + [5] + []
```

code : 5-11.py

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

```
   selection_sort([3,5,4,2])
=> [2] + selection_sort([3,5,4])
=> [2] + [3] + selection_sort([5,4])
=> [2] + [3] + [4] + selection_sort([5])
=> [2] + [3] + [4] + [5] + selection_sort([])
=> [2] + [3] + [4] + [5] + []
== [2,3,4,5]
```

재귀

code : 5-11.py

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

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2])
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) ==
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) ==
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=> loop([],[2,3,4]+[5]) ==
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=> loop([],[2,3,4]+[5]) == loop([],[2,3,4,5])
=>
```

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

```
   selection_sort([3,5,4,2])
=> loop([3,5,4,2],[])
=> loop([3,5,4],[]+[2]) == loop([3,5,4],[2])
=> loop([5,4],[2]+[3]) == loop([5,4],[2,3])
=> loop([5],[2,3]+[4]) == loop([5],[2,3,4])
=> loop([],[2,3,4]+[5]) == loop([],[2,3,4,5])
=> [2,3,4,5]
```

리스트 메소드

연산	의미	비고
xs.append(x)	xs의 맨 뒤에 x을 붙인다.	-

꼬리 재귀

code : 5-13.py

```
1  def selection_sort(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,[])
```

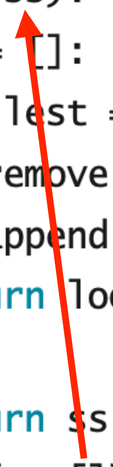
code : 5-14.py

```
1  def selection_sort(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,[])
```

꼬리 재귀

code : 5-14.py

```
1  def selection_sort(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,[])
```



while 루프

code : 5-15.py

```
1  def selection_sort(xs):
2      ss = []
3      while xs != []:
4          smallest = min(xs)
5          xs.remove(smallest)
6          ss.append(smallest)
7      return ss
```

꼬리 재귀

code : 5-14.py

```
1  def selection_sort(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,[])
```

while 루프

code : 5-15.py

```
1  def selection_sort(xs):
2      ss = []
3      while xs != []:
4          smallest = min(xs)
5          xs.remove(smallest)
6          ss.append(smallest)
7      return ss
```

꼬리 재귀

code : 5-14.py

```
1  def selection_sort(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,[])
```

while 루프

code : 5-15.py

```
1  def selection_sort(xs):
2      ss = []
3      while xs != []:
4          smallest = min(xs)
5          xs.remove(smallest)
6          ss.append(smallest)
7      return ss
```

꼬리 재귀

code : 5-14.py

```
1  def selection_sort(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,[])
```

while 루프

code : 5-15.py

```
1  def selection_sort(xs):
2      ss = []
3      while xs != []:
4          smallest = min(xs)
5          xs.remove(smallest)
6          ss.append(smallest)
7      return ss
```

삽입정렬

Insertion Sort

삽입정렬 알고리즘
Insertion Sort

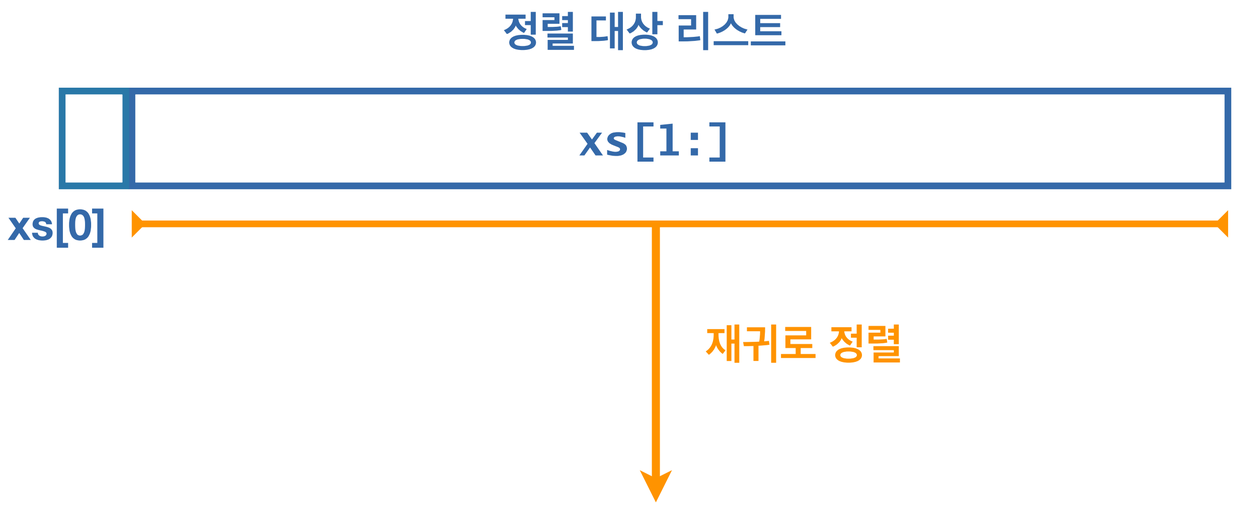
리스트 xs를 정렬하려면		
반복 조건	`xs != []`	• xs의 후미리스트인 `xs[1:]`를 재귀로 정렬하고, • xs의 선두원소인 `xs[0]`를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 `[]`를 그대로 리턴한다.

리스트 xs를 정렬하려면		
반복 조건	`xs != []`	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

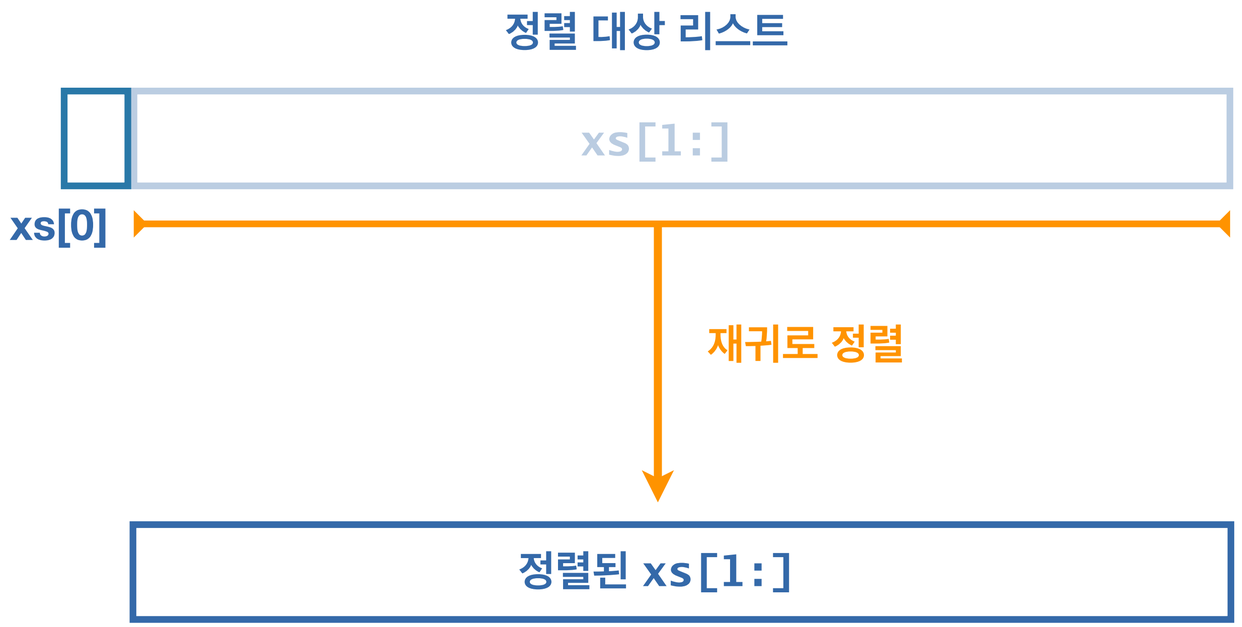
리스트 xs를 정렬하려면		
반복 조건	xs != []	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

정렬 대상 리스트

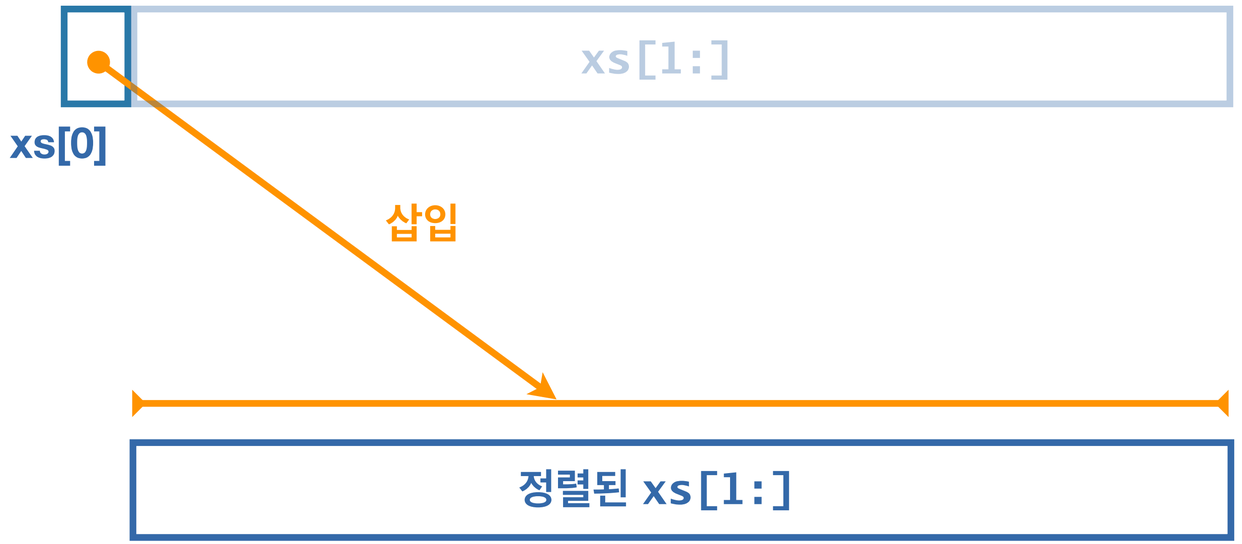
리스트 xs를 정렬하려면		
반복 조건	`xs != []`	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	`xs == []`	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



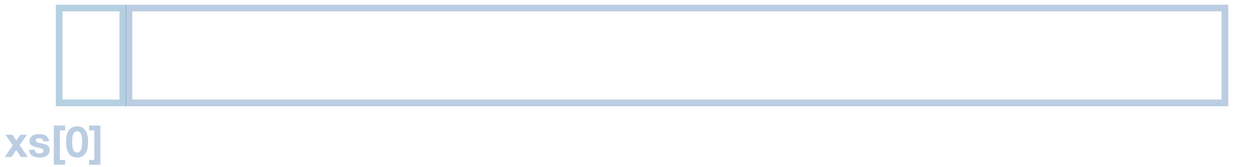
리스트 xs를 정렬하려면		
반복 조건	xs != []	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬하려면		
반복 조건	xs != []	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



리스트 xs를 정렬하려면		
반복 조건	xs != []	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.



정렬된 xs

리스트 xs를 정렬하려면		
반복 조건	xs != []	• xs의 후미리스트인 xs[1:]를 재귀로 정렬하고, • xs의 선두원소인 xs[0]를 정렬된 리스트의 적절한 위치에 끼워서 리턴한다.
종료 조건	xs == []	• 정렬할 원소가 없으므로 []를 그대로 리턴한다.

code : 5-16.py

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

code : 5-16.py

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

```
insertionsort([3,5,4,2])
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
=> insert(3,insert(5,[2,4]))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
=> insert(3,insert(5,[2,4]))
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
=> insert(3,insert(5,[2,4]))
=> insert(3,[2,4,5])
```

code : 5-16.py

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

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
=> insert(3,insert(5,[2,4]))
=> insert(3,[2,4,5])
```

code : 5-16.py

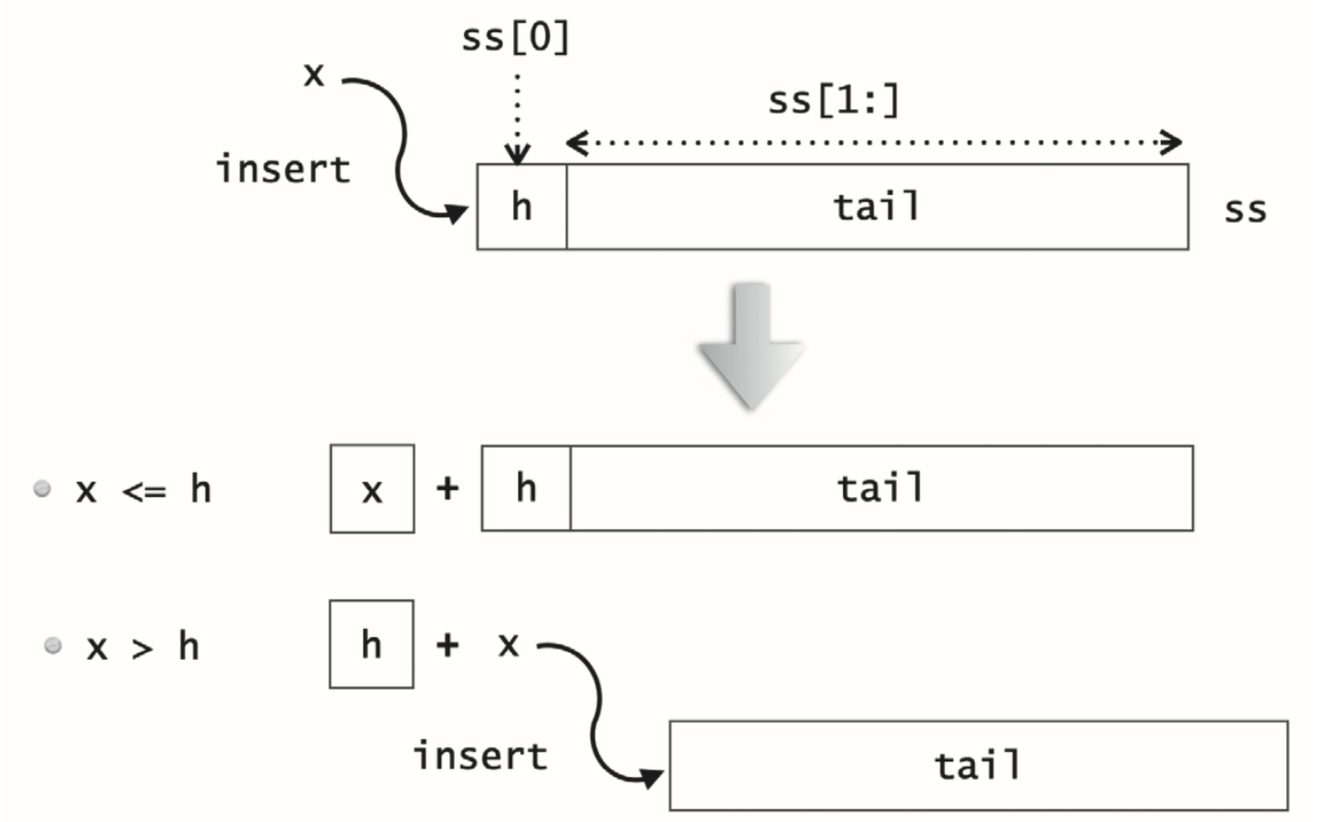
```
1 def insertion_sort(xs):
2     if xs != []:
3         return insert(xs[0],insertion_sort(xs[1:]))
4     else:
5         return []
```

```
   insertionsort([3,5,4,2])
=> insert(3,insertionsort([5,4,2]))
=> insert(3,insert(5,insertionsort([4,2])))
=> insert(3,insert(5,insert(4,insertionsort([2]))))
=> insert(3,insert(5,insert(4,insert(2,insertionsort([])))))
=> insert(3,insert(5,insert(4,insert(2,[]))))
=> insert(3,insert(5,insert(4,[2])))
=> insert(3,insert(5,[2,4]))
=> insert(3,[2,4,5])
=> [2,3,4,5]
```

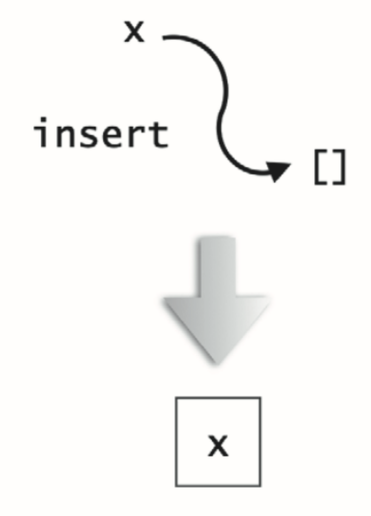
insert(x,ss)

원소 x를 정렬된 리스트 ss의 적절한 위치에 끼워넣기

`ss != []`



`ss == []`



`[x]`

insert 알고리즘

정수 x을 정렬된 리스트 ss의 제 위치에 끼워 넣으려면,		
반복 조건	ss != []	• x <= ss[0]이면, x를 ss의 앞에 붙여서 리턴한다. • x > ss[0]이면, 재귀로 x를 ss[1:]의 제 위치에 끼워넣고, 그 앞에 ss[0]를 붙여서 리턴한다.
종료 조건	ss == []	그냥 x만 가지고 리스트를 만든다.

insert 실행추적 사례

```
insert(9,[])
```

insert 실행추적 사례

```
   insert(9,[])
=> [9]
```

insert 실행추적 사례

```
insert(1,[2,4,5,7,8])
```

insert 실행추적 사례

```
   insert(1,[2,4,5,7,8])
=> [1, 2, 4, 5, 7, 8]
```

insert 실행추적 사례

```
insert(6,[2,4,5,7,8])
```

insert 실행추적 사례

```
   insert(6,[2,4,5,7,8])
=> [2] + insert(6,[4,5,7,8])
```

insert 실행추적 사례

```
   insert(6,[2,4,5,7,8])
=> [2] + insert(6,[4,5,7,8])
=> [2] + [4] + insert(6,[5,7,8])
```

insert 실행추적 사례

```
   insert(6,[2,4,5,7,8])
=> [2] + insert(6,[4,5,7,8])
=> [2] + [4] + insert(6,[5,7,8])
=> [2] + [4] + [5] + insert(6,[7,8])
```

insert 실행추적 사례

```
   insert(6,[2,4,5,7,8])
=> [2] + insert(6,[4,5,7,8])
=> [2] + [4] + insert(6,[5,7,8])
=> [2] + [4] + [5] + insert(6,[7,8])
=> [2] + [4] + [5] + [6] + [7,8]
```

insert 실행추적 사례

```
   insert(6,[2,4,5,7,8])
=> [2] + insert(6,[4,5,7,8])
=> [2] + [4] + insert(6,[5,7,8])
=> [2] + [4] + [5] + insert(6,[7,8])
=> [2] + [4] + [5] + [6] + [7,8]
   . . .
=> [2,4,5,6,7,8]
```

pp.233~236

실습 5.2 insert : 재귀 함수 버전

이제 알고리즘에 기술한 대로 insert 재귀 함수를 다음 뼈대코드에 맞추어 구현하자.

code : 5-17.py

```
1  def insert(x,ss):
2      if ss != []:
3          if x <= ss[0]:
4              pass # Write your code here.
5          else:
6              pass # Write your code here.
7      else:
8          return [x]
```

```
insert(1,[2,4,5,7,8])
insert(6,[2,4,5,7,8])
insert(9,[])
```

실습 5.3 insert : 꼬리재귀 함수 버전

〈실습 5.2〉에서 완성한 재귀 함수 insert는 꼬리 재귀가 아니다. append 메소드를 활용하여 꼬리재귀 함수를 다음 뼈대코드에 맞추어 완성하자.

code : 5-18.py

```
1  def insert(x,ss):
2      def loop(ss,left):
3          if ss != []:
4              if x <= ss[0]:
5                  pass # Write your code here.
6
7              else:
8                  pass # Write your code here.
9
10         else:
11             left.append(x)
12             return left
13     return loop(ss,[])
```

```
insert(1,[2,4,5,7,8])
insert(6,[2,4,5,7,8])
insert(9,[])
```

실습 5.4 insert : while 루프 버전

〈실습 5.3〉에서 작성한 꼬리재귀 함수를 참고하여, insert 함수의 while 루프 버전을 다음 뼈대코드에 맞추어 완성하자.

code : 5-19.py

```
1  def insert(x,ss):
2      left = []
3      while ss != []:
4          if x <= ss[0]:
5              pass # Write your code here.
6
7          else:
8              pass # Write your code here.
9
10     left.append(x)
11     return left
```

삽입정렬 함수

실습 5.5 insertion_sort : 꼬리재귀 함수 버전

insert 함수를 완성했으므로, 이제 다음 삽입정렬 재귀 함수 insertion_sort가 작동한다.

code : 5-20.py

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

그런데 이 재귀 함수는 꼬리재귀가 아니다. 다음 뼈대코드에 맞추어 꼬리재귀 함수로 만들자.

code : 5-21.py

```
1 def insertion_sort(xs):
2     def loop(xs,ss):
3         if xs != []:
4             pass # Write your code here.
5         else:
6             pass # Write your code here.
7     return loop(xs,[])
```

```
insertion_sort(3,5,4,2])
```

실습 5.6 insertion_sort : while 루프 버전

〈실습 5.5〉에서 작성한 꼬리재귀 함수를 참고하여, insertion_sort 함수의 while 루프 버전을 다음 뼈대코드에 맞추어 완성하자.

code : 5-22.py

```
1  def insertion_sort(xs) :
2      ss = []
3      while xs != []:
4          xs, ss = None, None # Write your expressions here.
5      return ss
```

실습 5.7 insertion_sort : for 루프 버전

이번에는 insertion_sort 함수를 for 루프로 다음 뼈대코드에 맞추어 작성하자.

code : 5-23.py

```
1  def insertion_sort(xs):
2      ss = []
3      pass # Write your for-loop here.
4
5      return ss
```

합병정렬

Merge Sort

합병정렬 알고리즘
Merge Sort

리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

정렬 대상 리스트

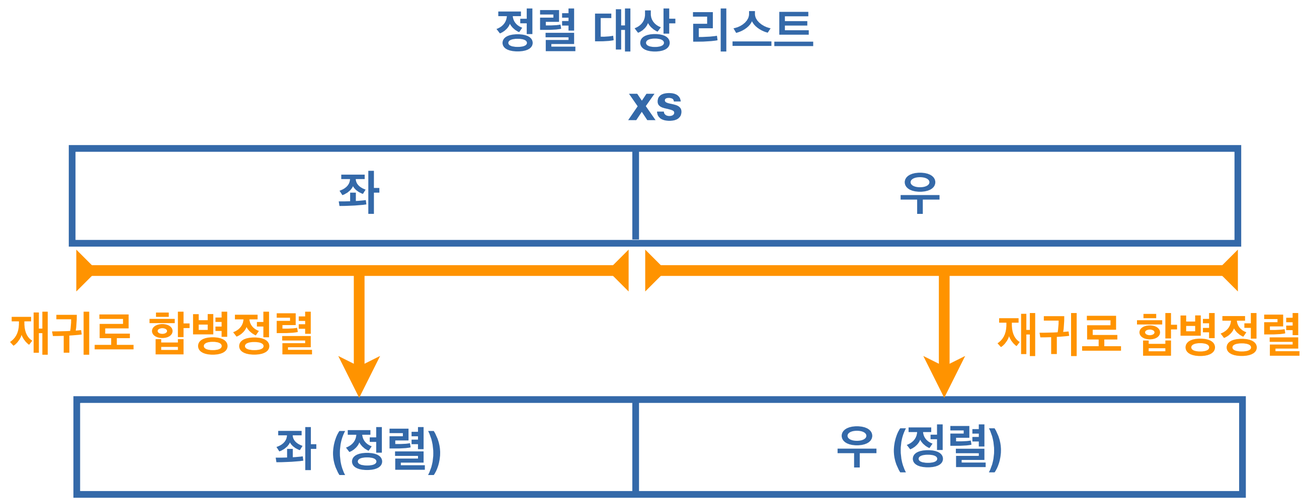
xs

리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

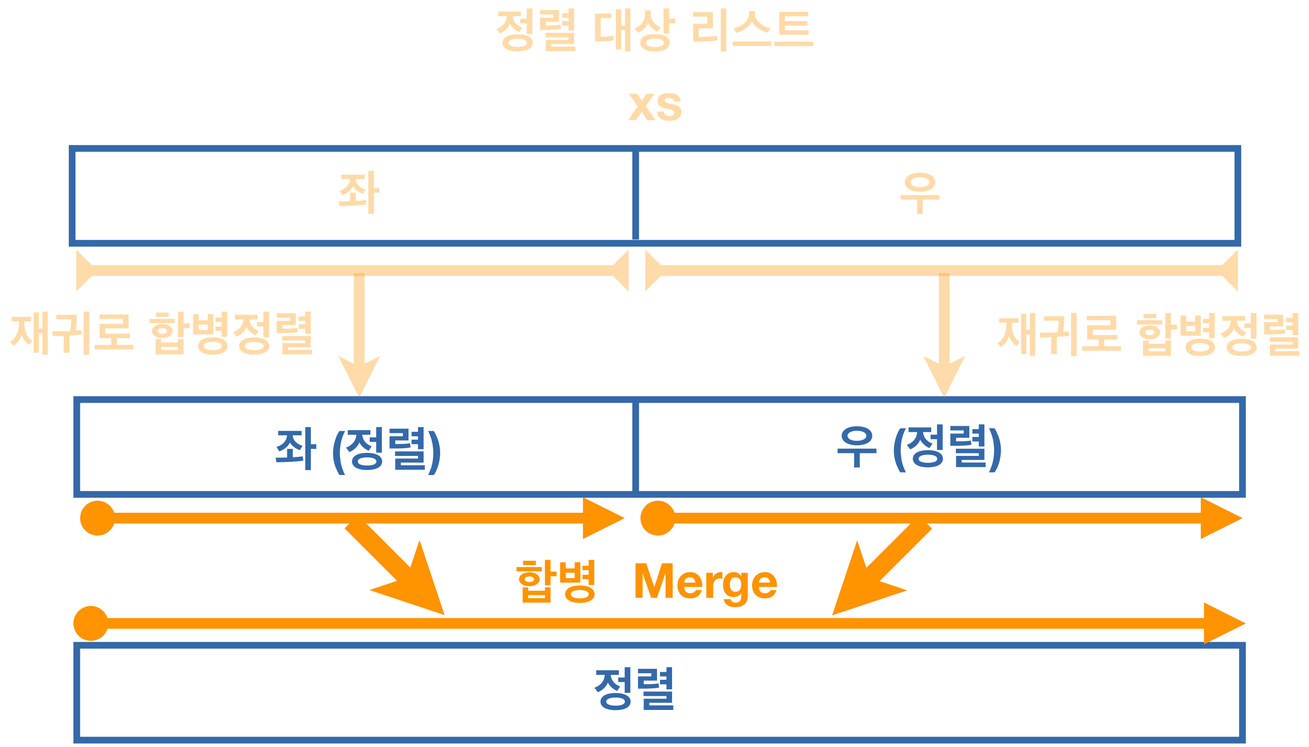
정렬 대상 리스트

xs

리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

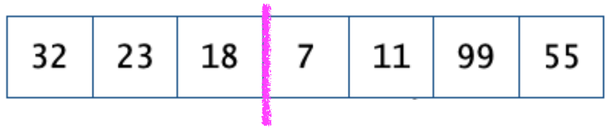


리스트 xs를 합병정렬하려면		
반복 조건	`len(xs) > 1`	• xs를 반으로 나누어, 각각 재귀로 합병정렬 완료하고, • 정렬된 두 리스트를 앞에서부터 차례로 훑어가며 • 두 선두원소 중에서 작은 원소를 먼저 하나씩 취하는 방식으로 하나로 합병merge하여 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.



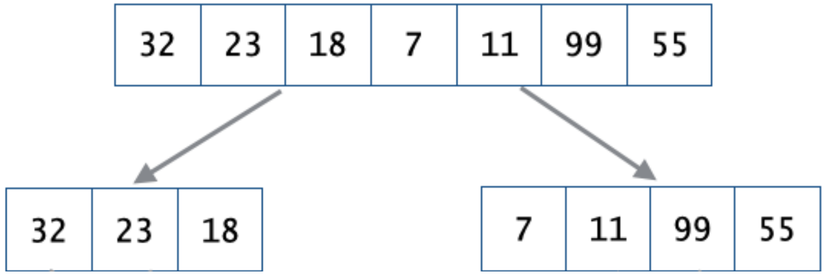
합병정렬

Merge Sort



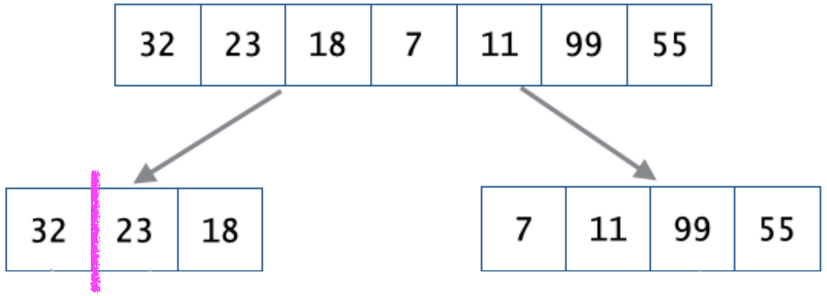
합병정렬

Merge Sort



합병정렬

Merge Sort

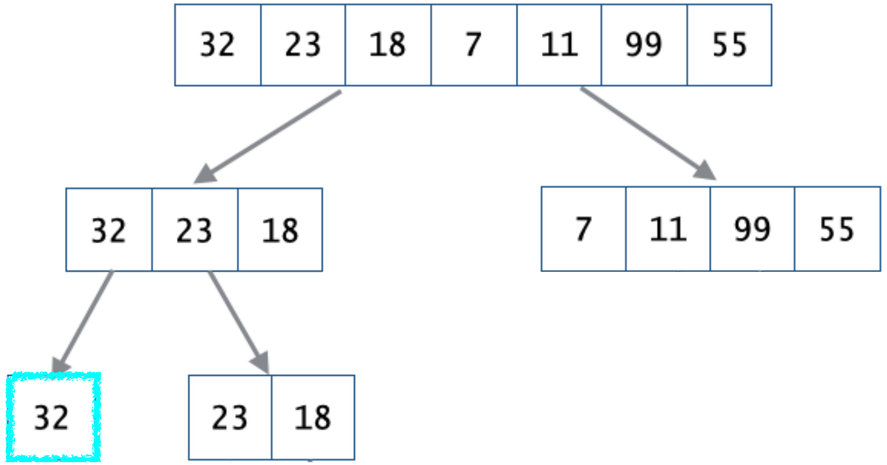


합병정렬

Merge Sort

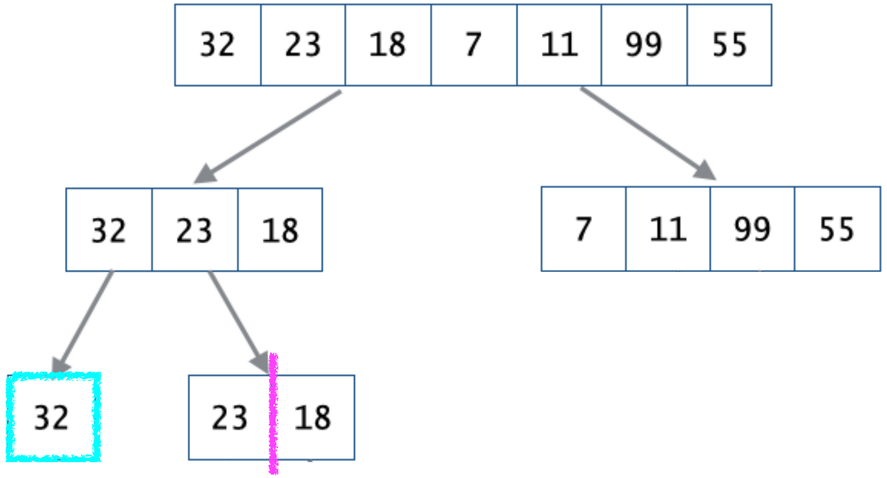
합병정렬

Merge Sort



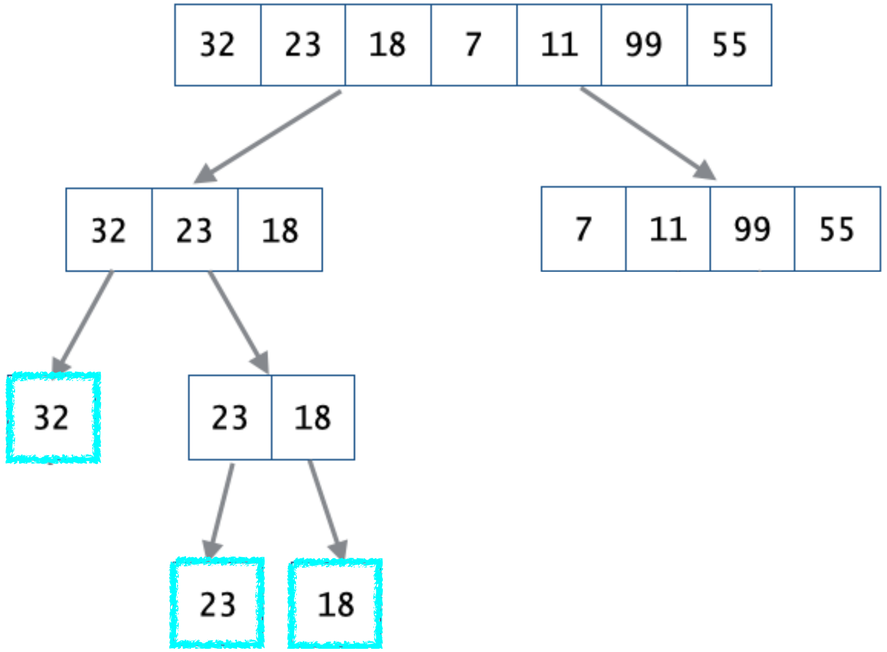
합병정렬

Merge Sort



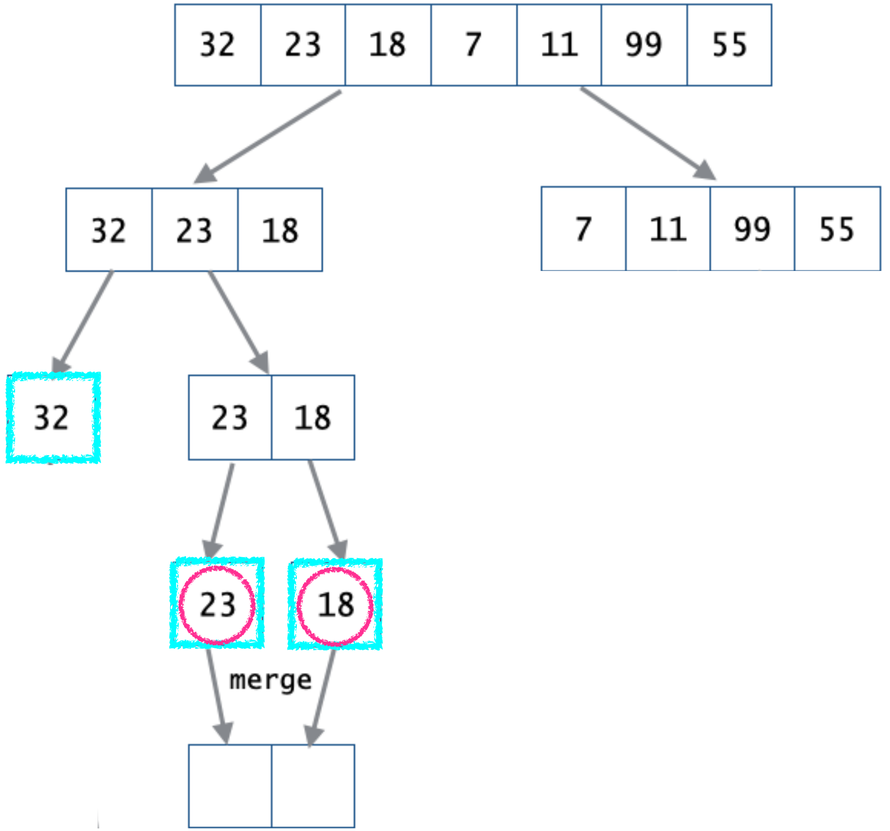
합병정렬

Merge Sort



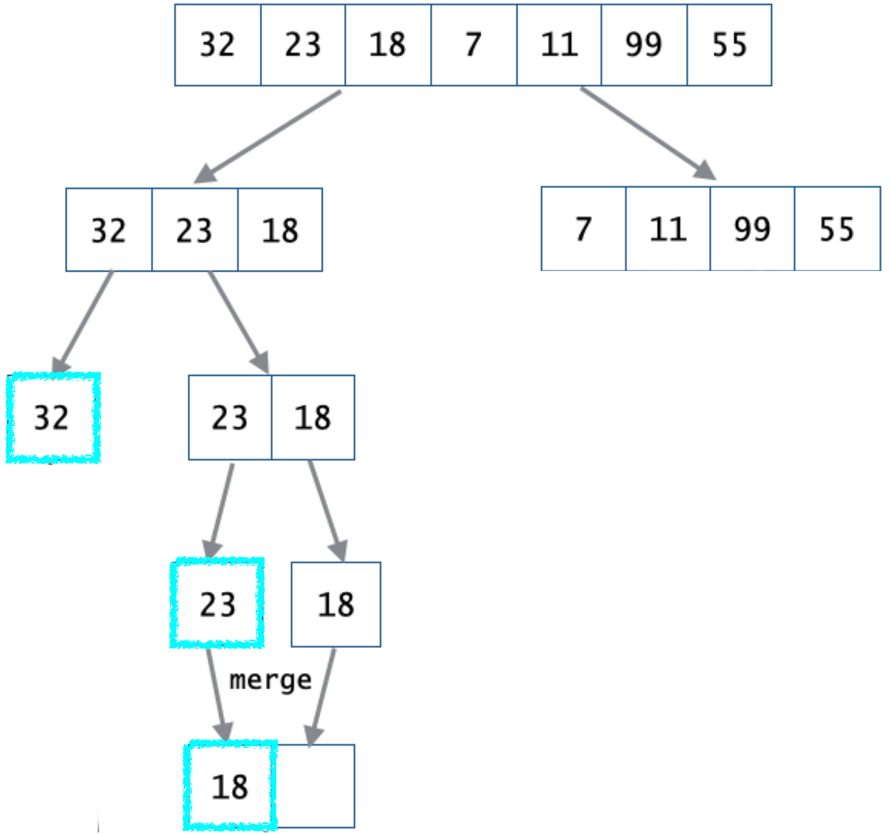
합병정렬

Merge Sort



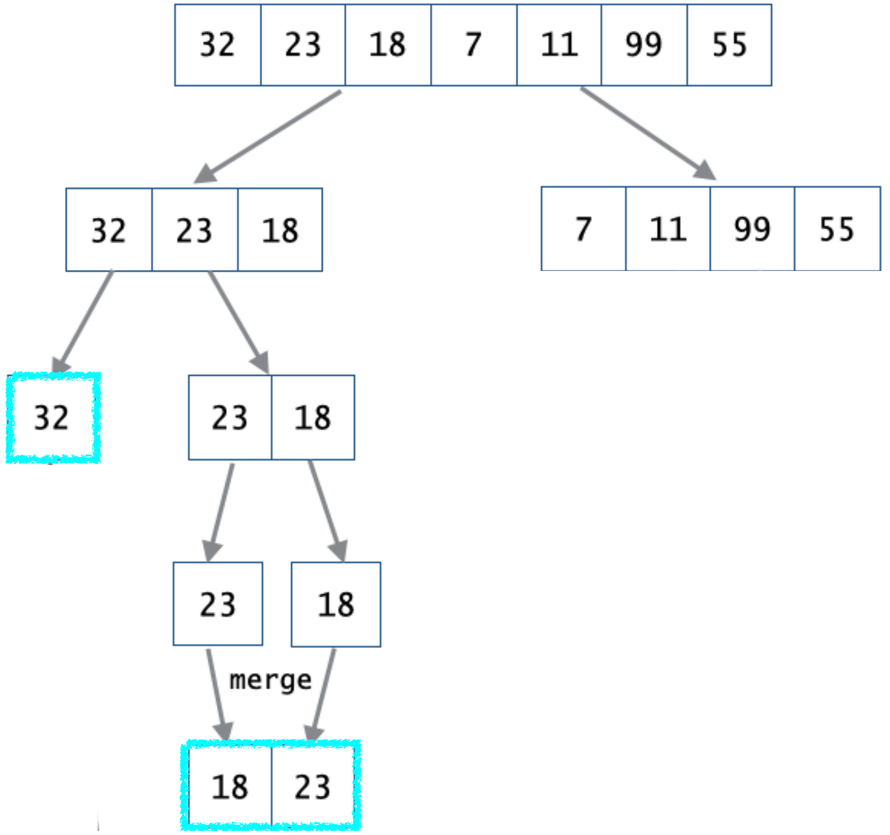
합병정렬

Merge Sort



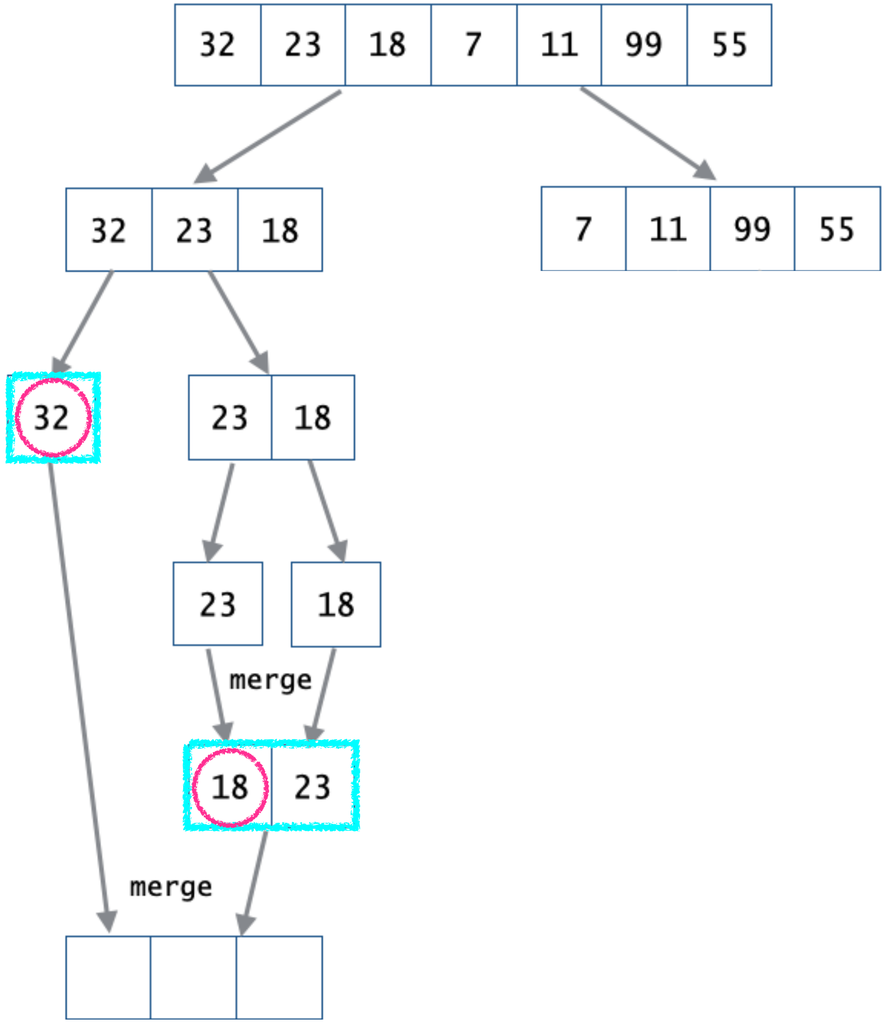
합병정렬

Merge Sort



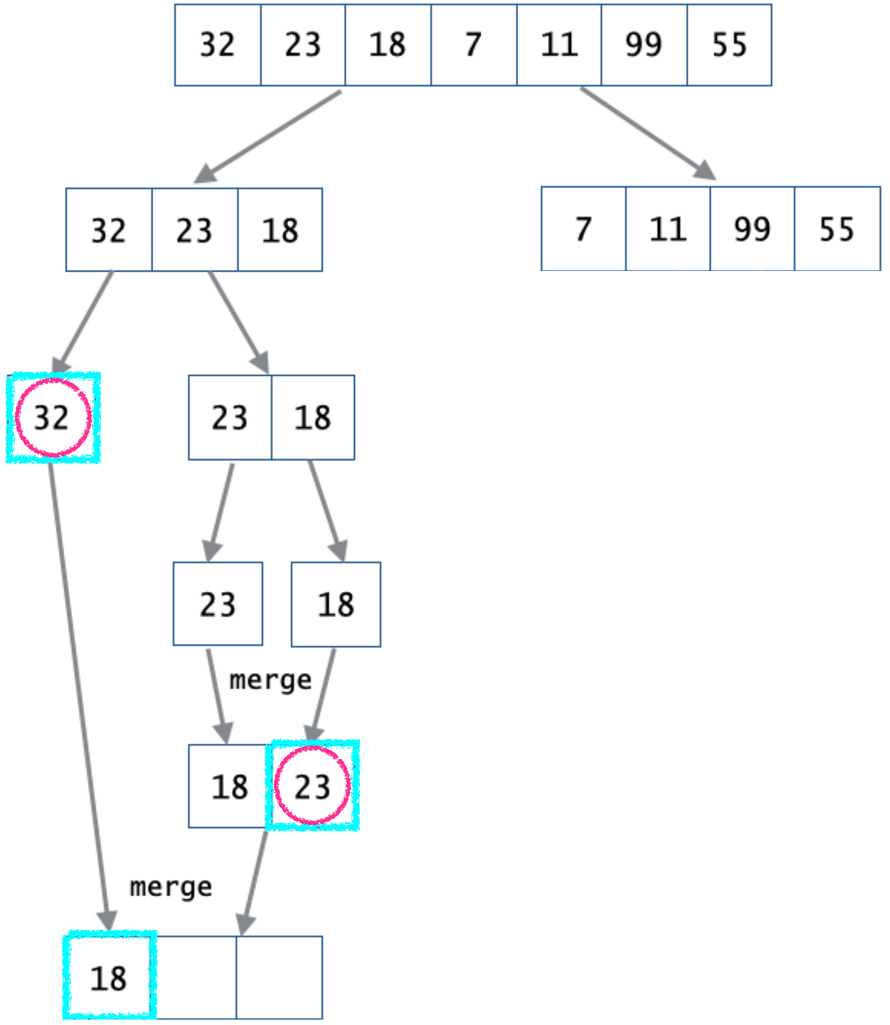
합병정렬

Merge Sort



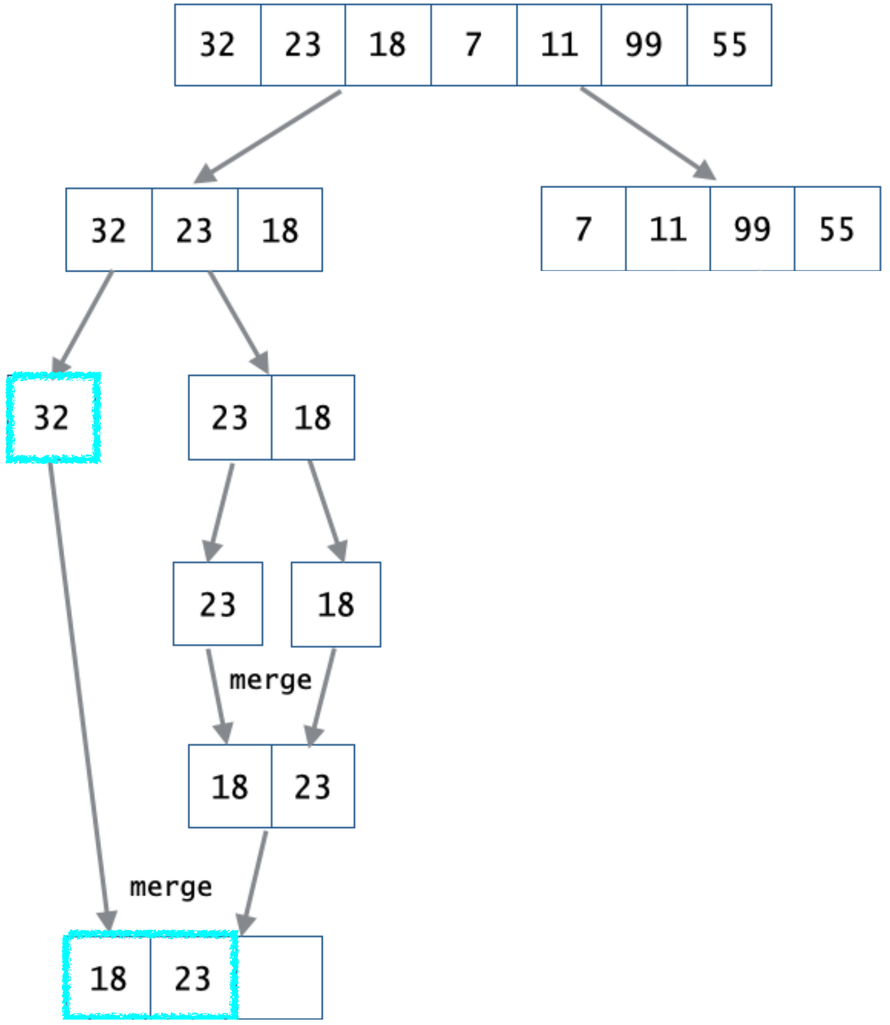
합병정렬

Merge Sort



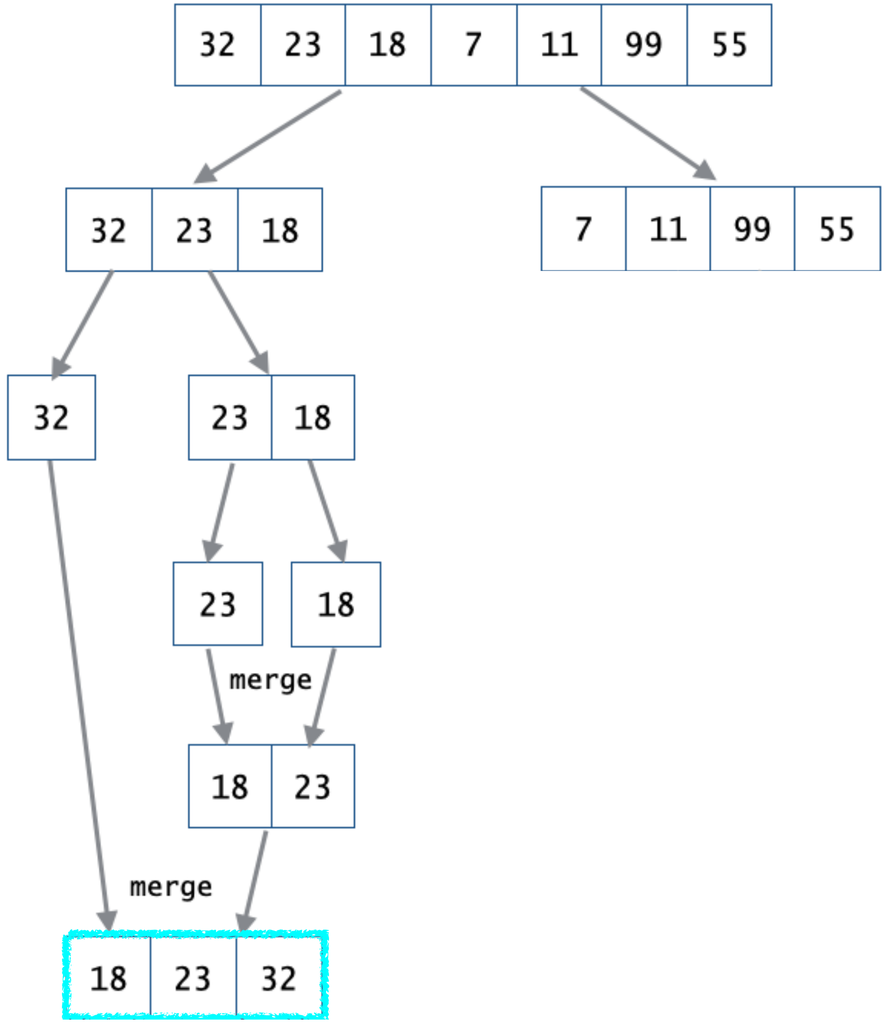
합병정렬

Merge Sort



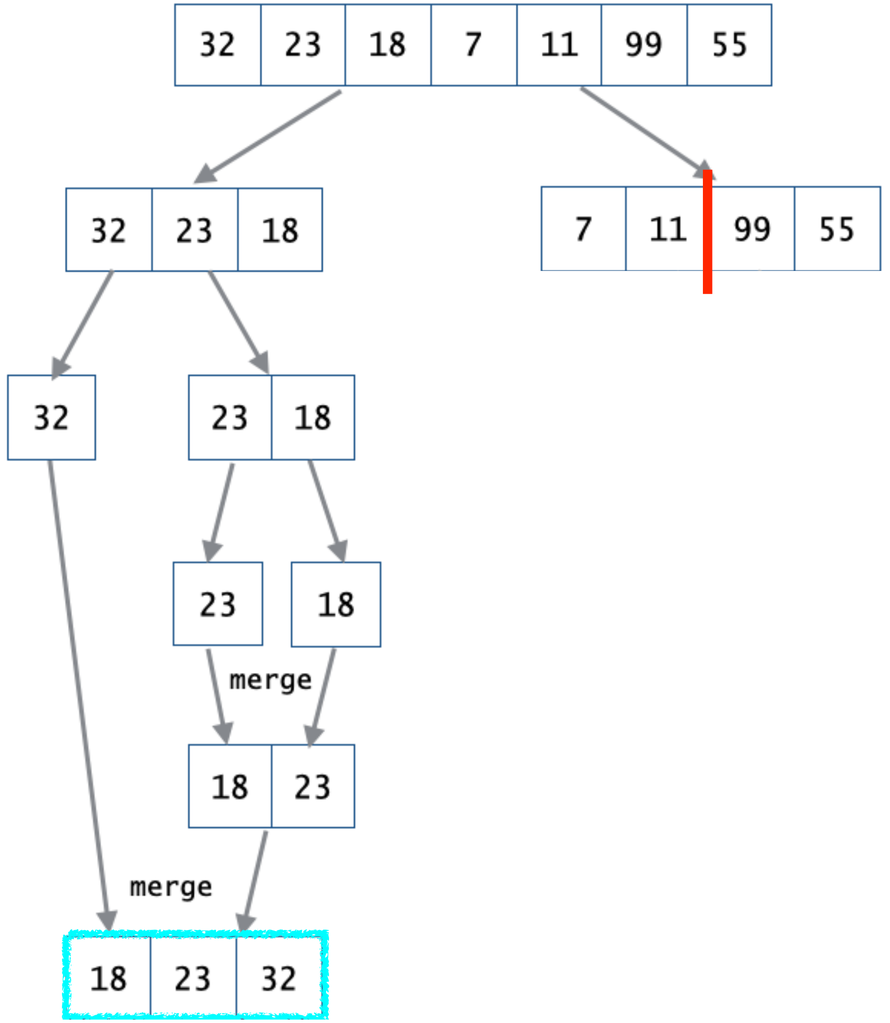
합병정렬

Merge Sort



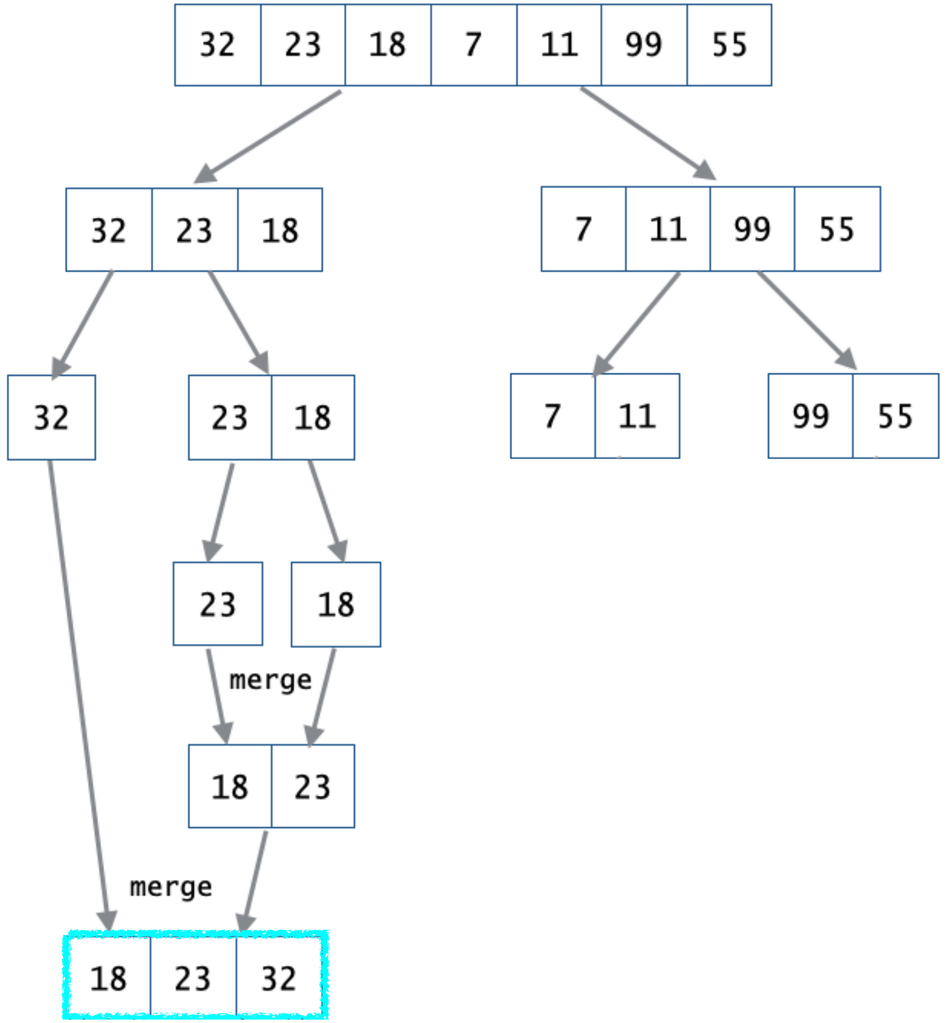
합병정렬

Merge Sort



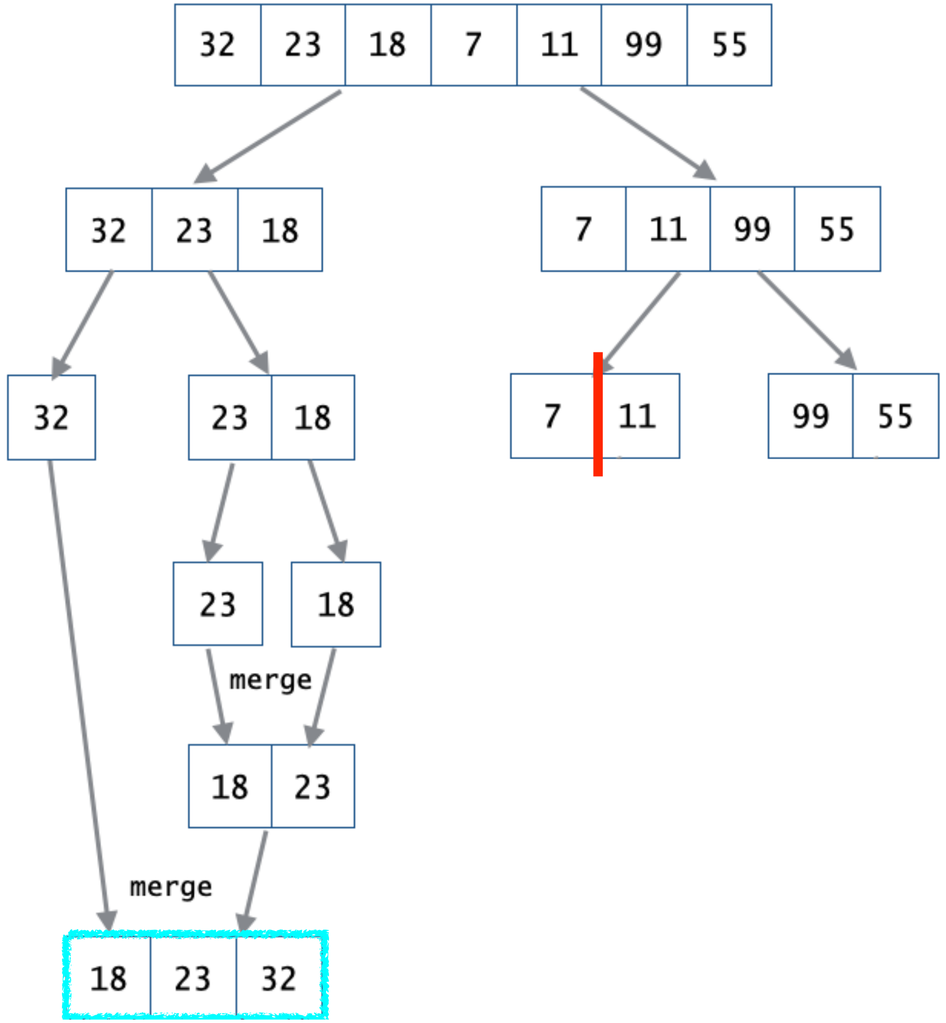
합병정렬

Merge Sort



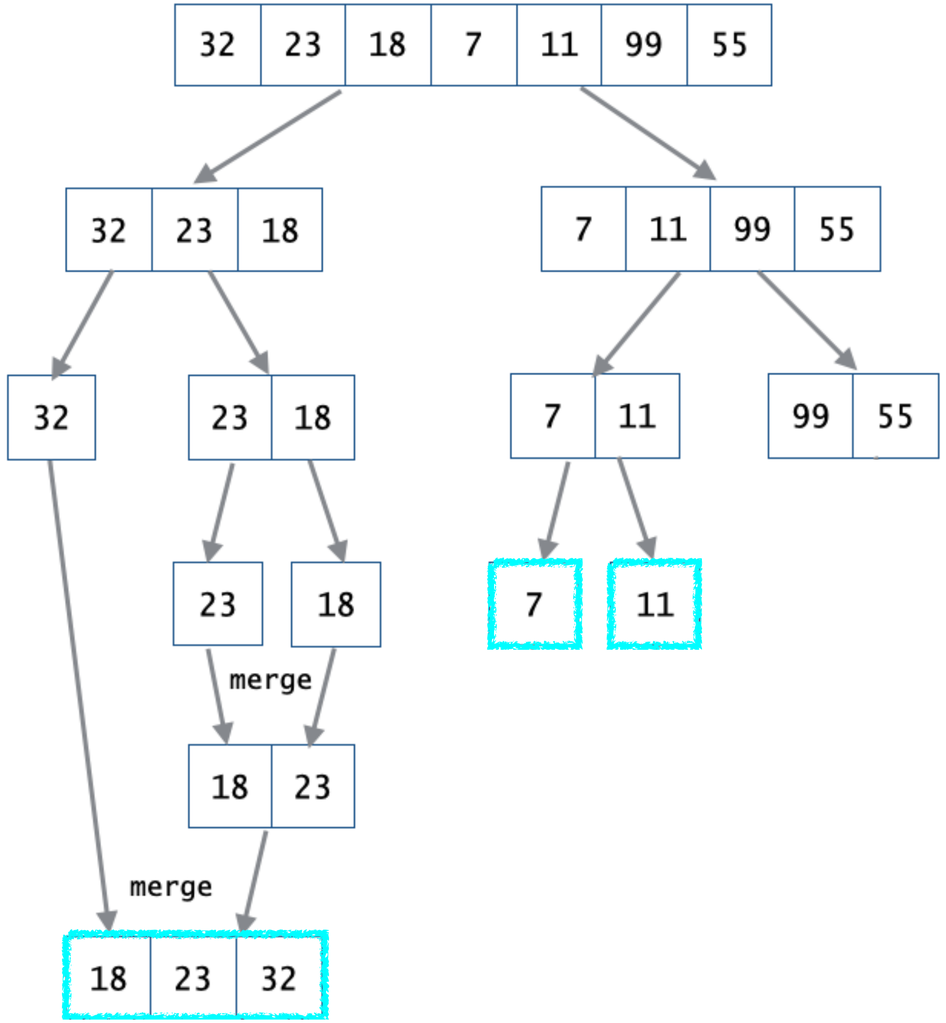
합병정렬

Merge Sort



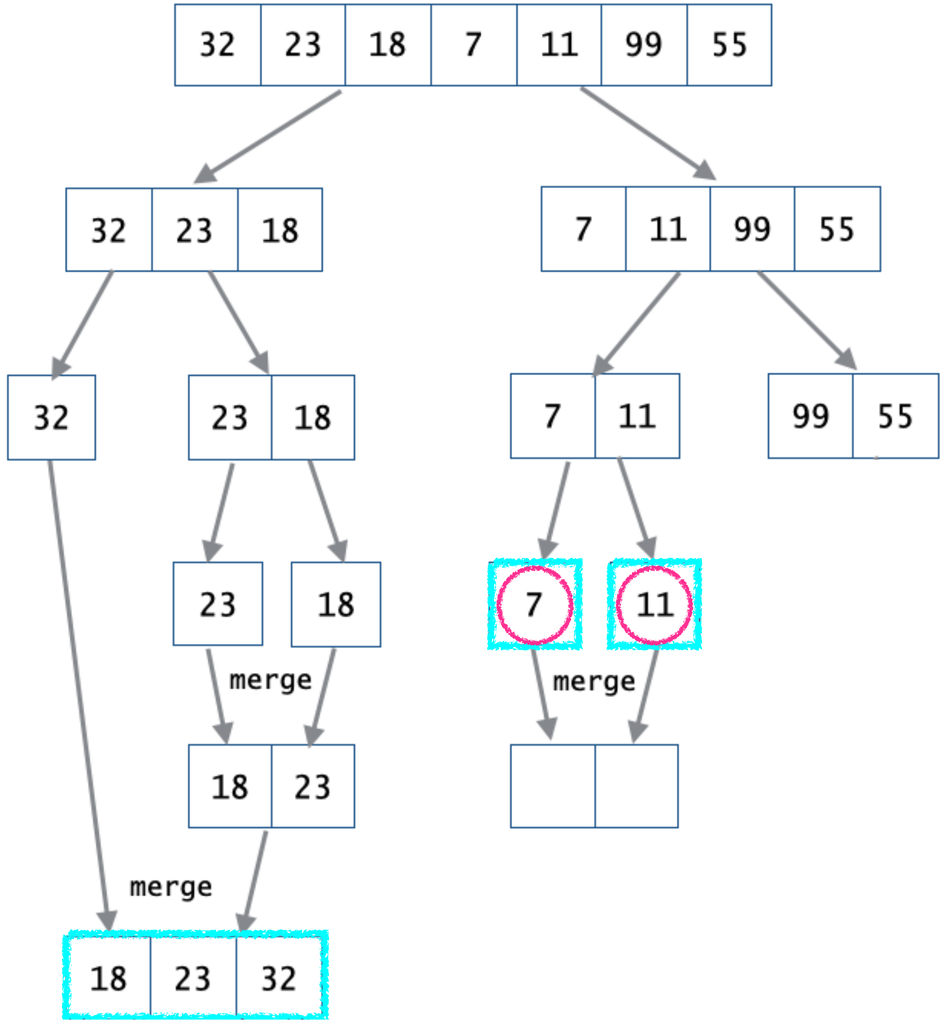
합병정렬

Merge Sort



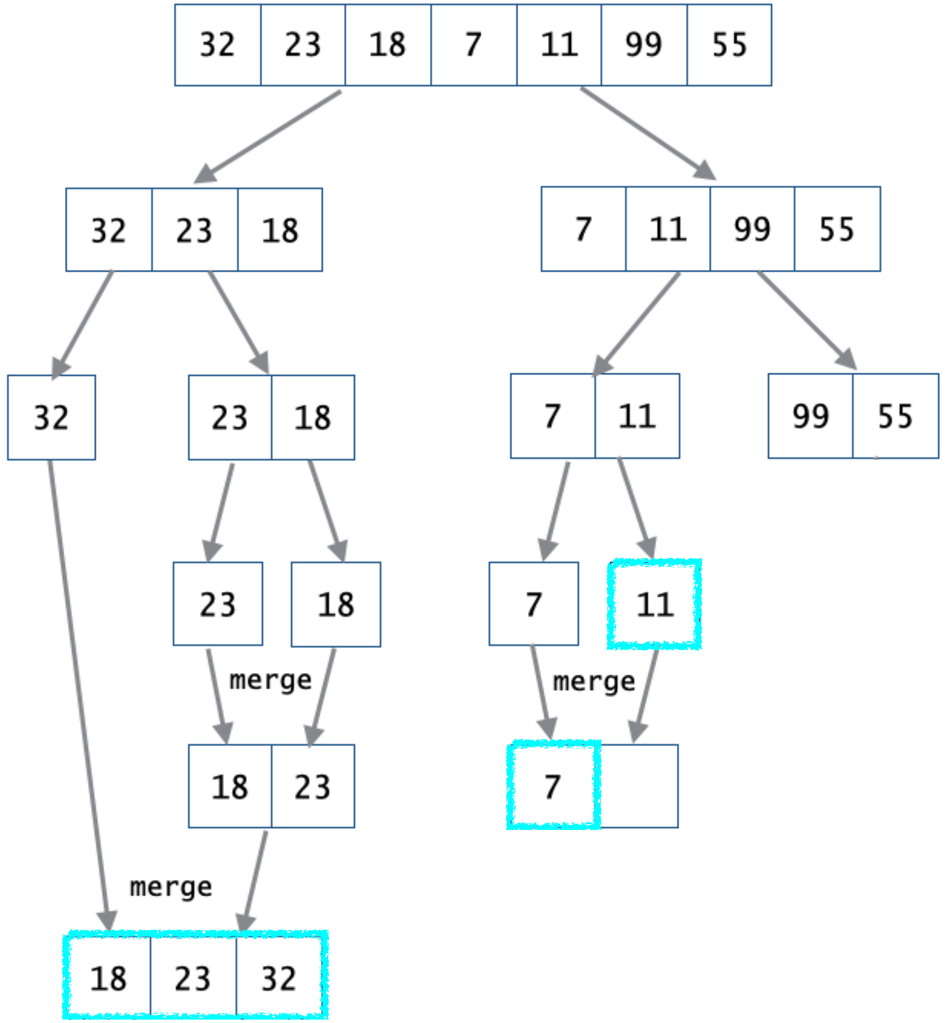
합병정렬

Merge Sort



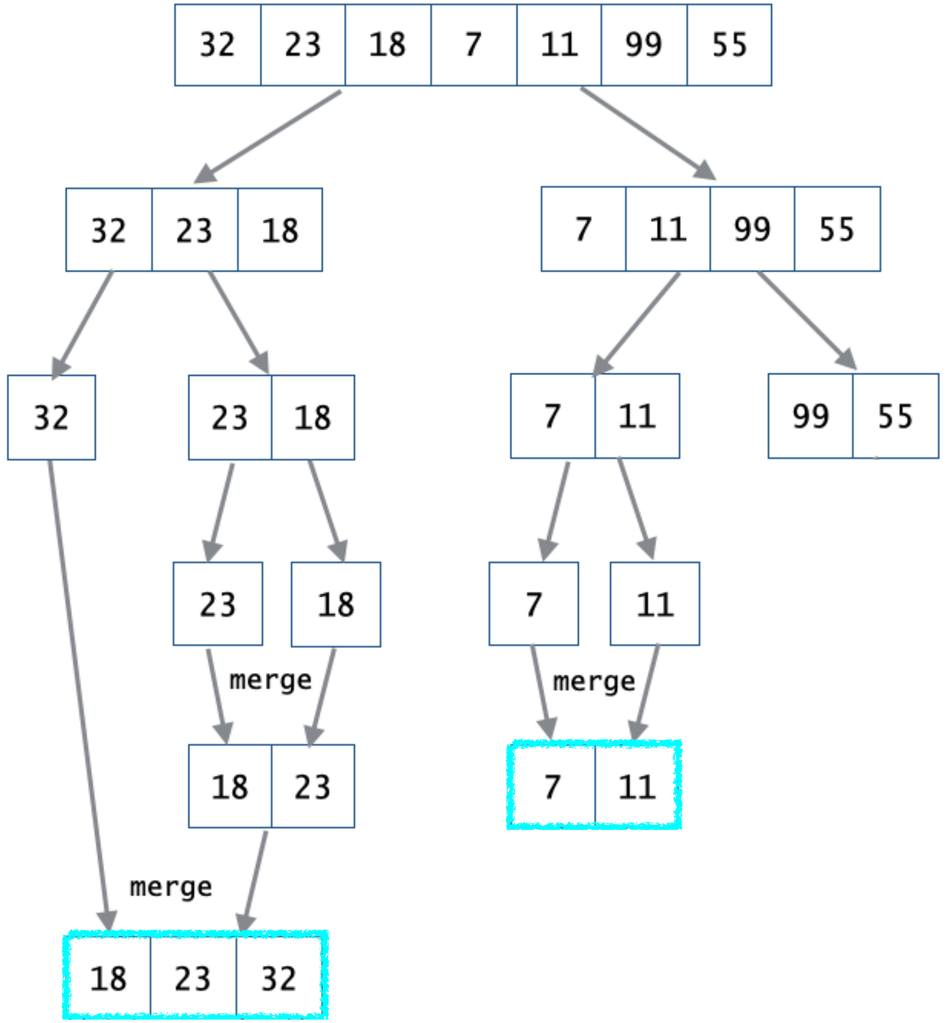
합병정렬

Merge Sort



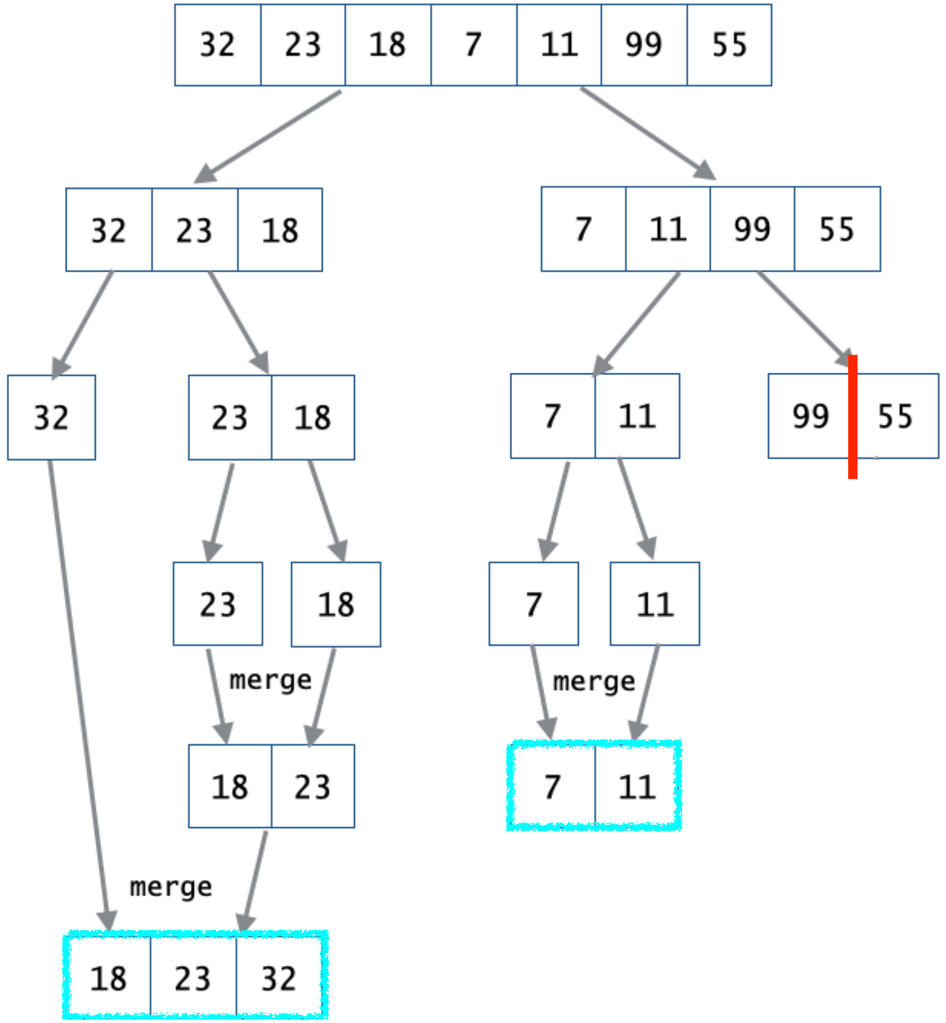
합병정렬

Merge Sort



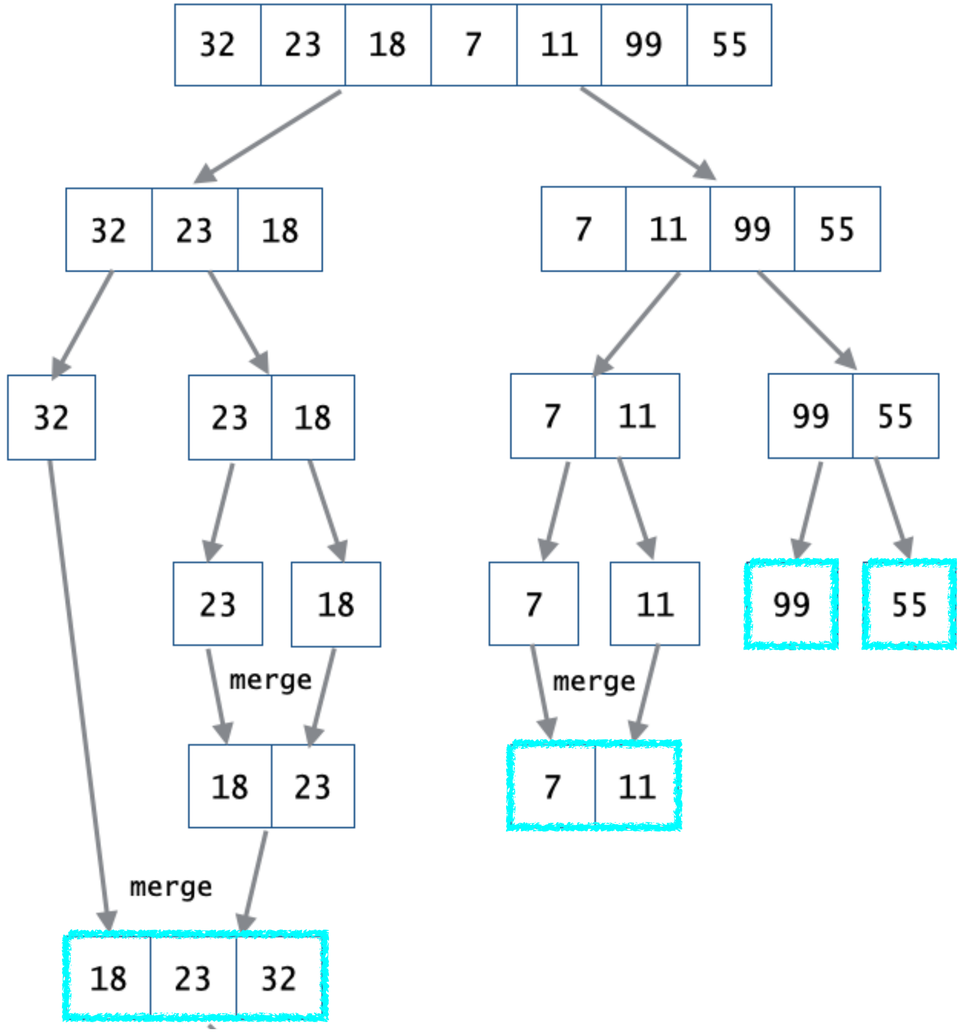
합병정렬

Merge Sort



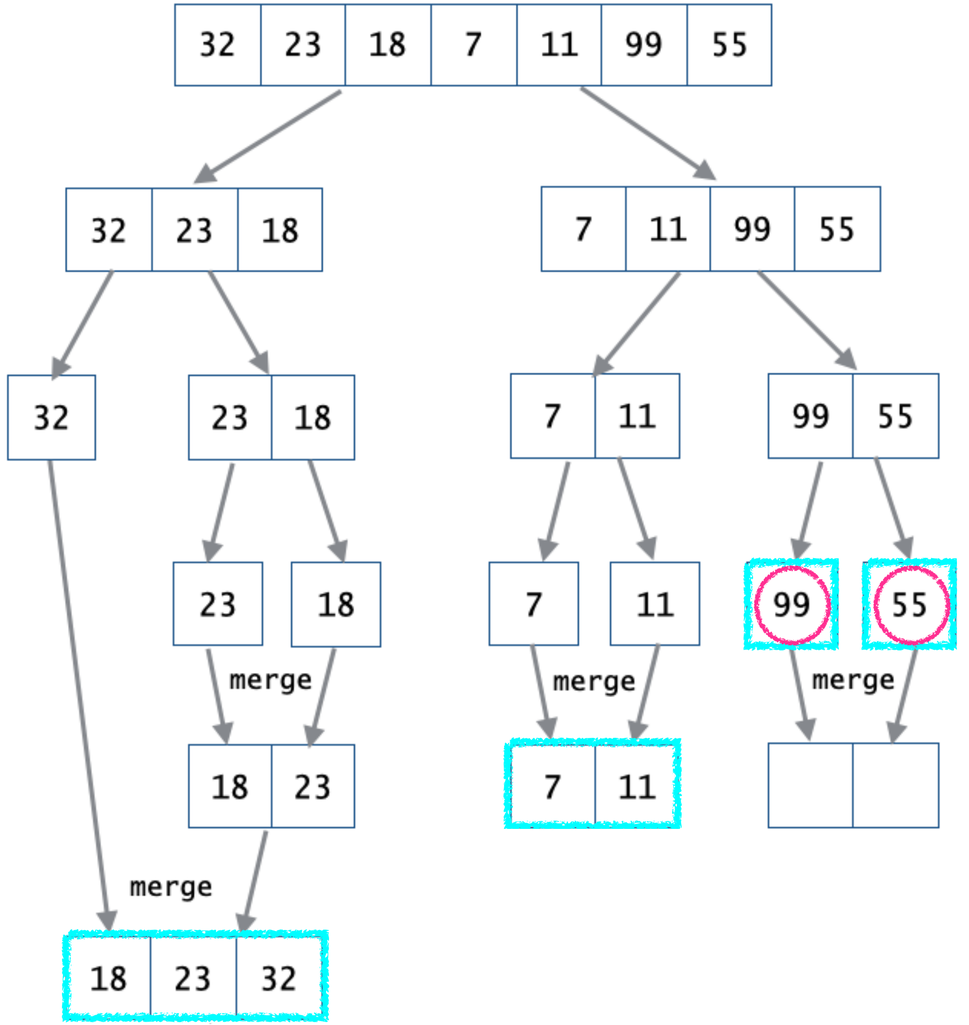
합병정렬

Merge Sort



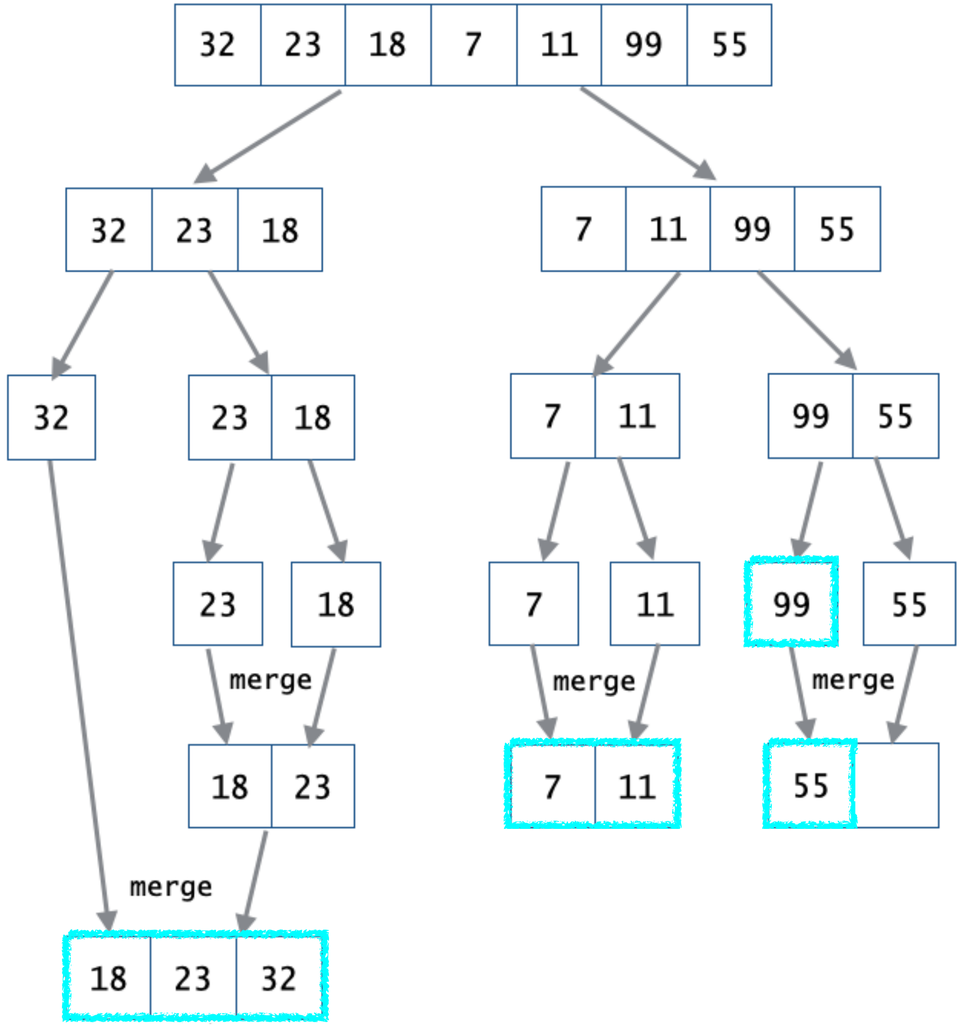
합병정렬

Merge Sort



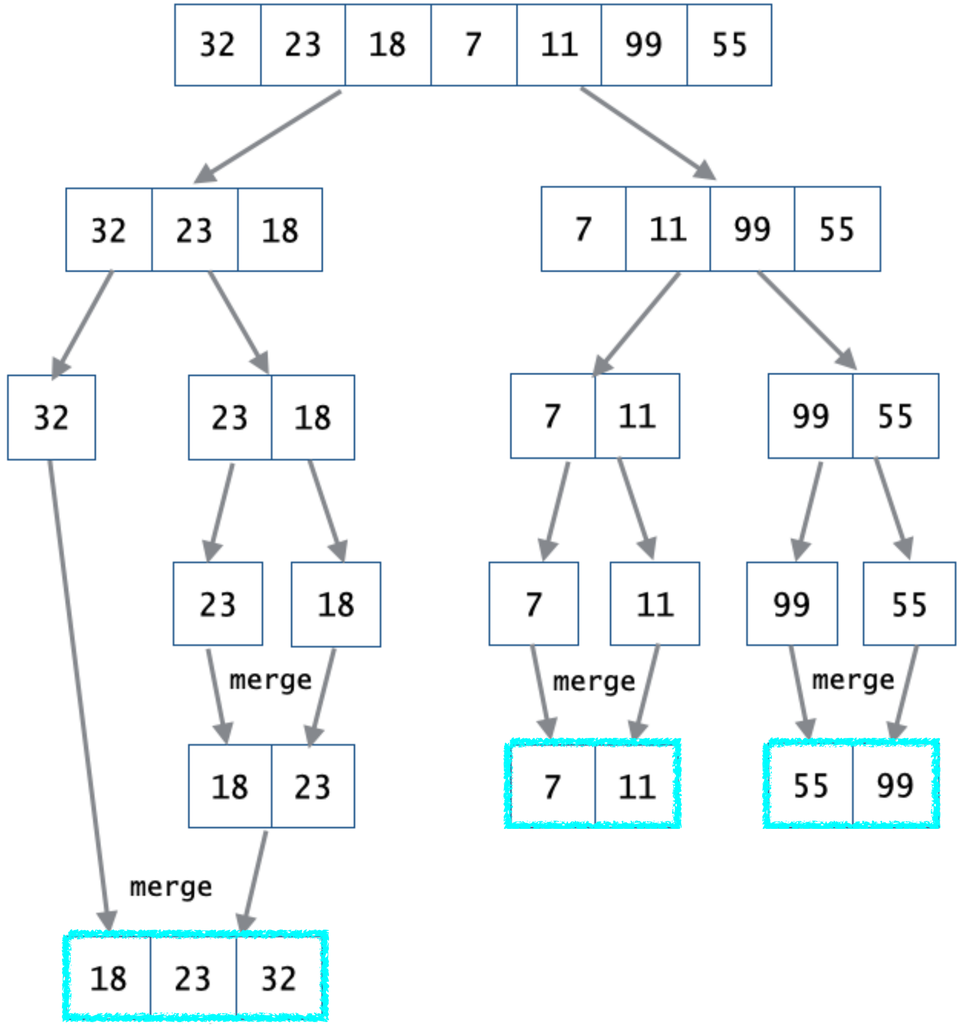
합병정렬

Merge Sort



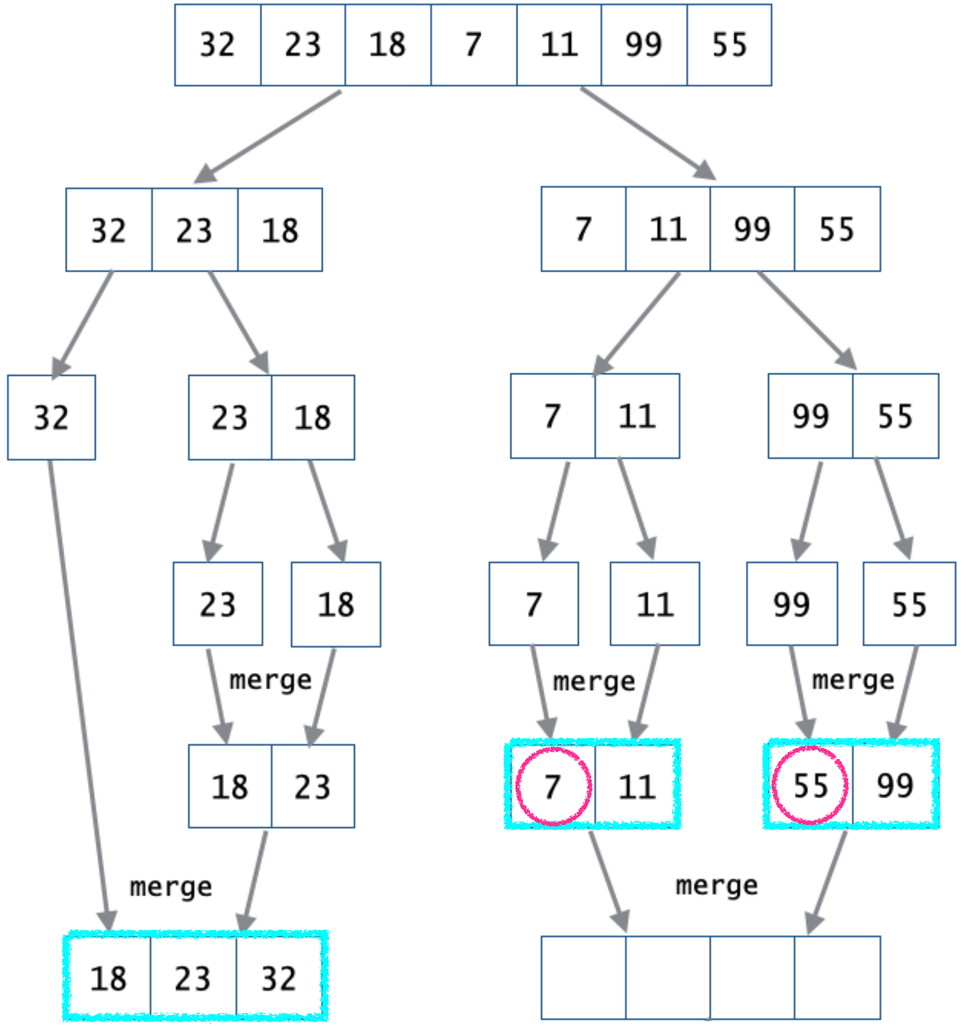
합병정렬

Merge Sort



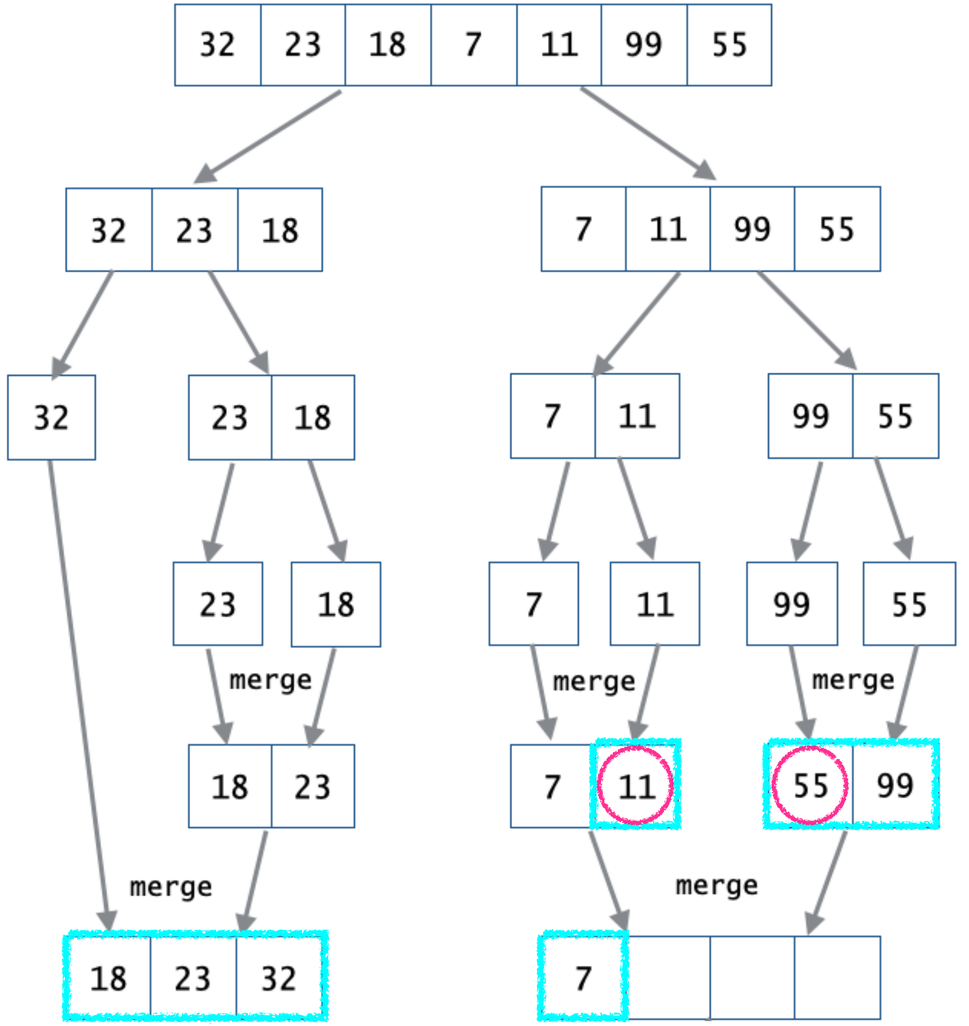
합병정렬

Merge Sort



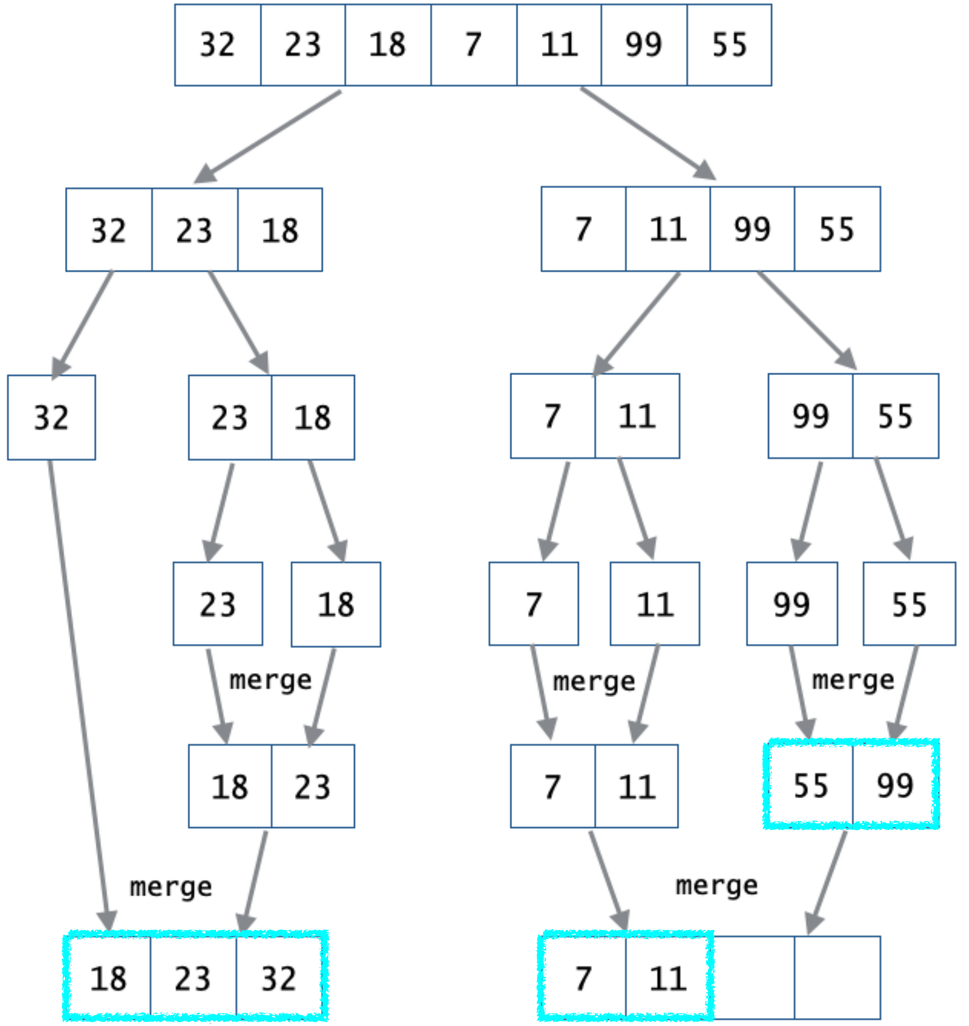
합병정렬

Merge Sort



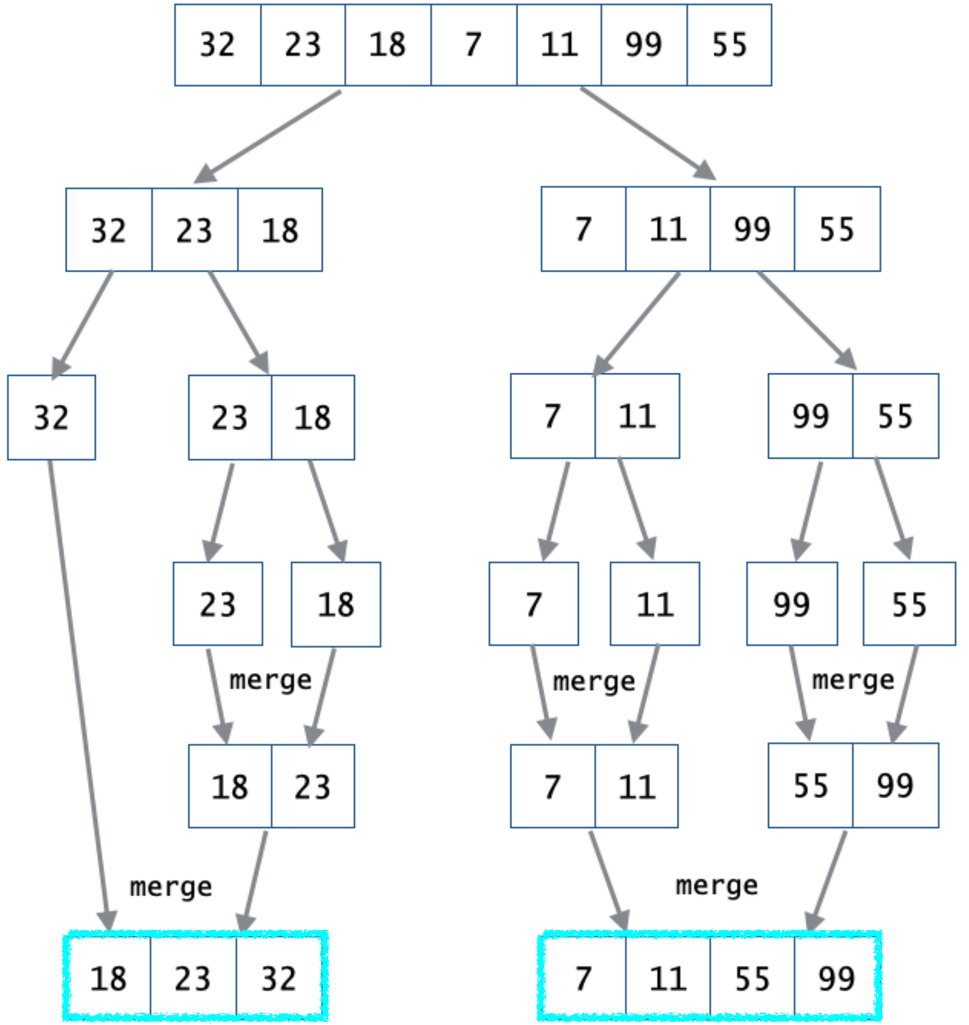
합병정렬

Merge Sort



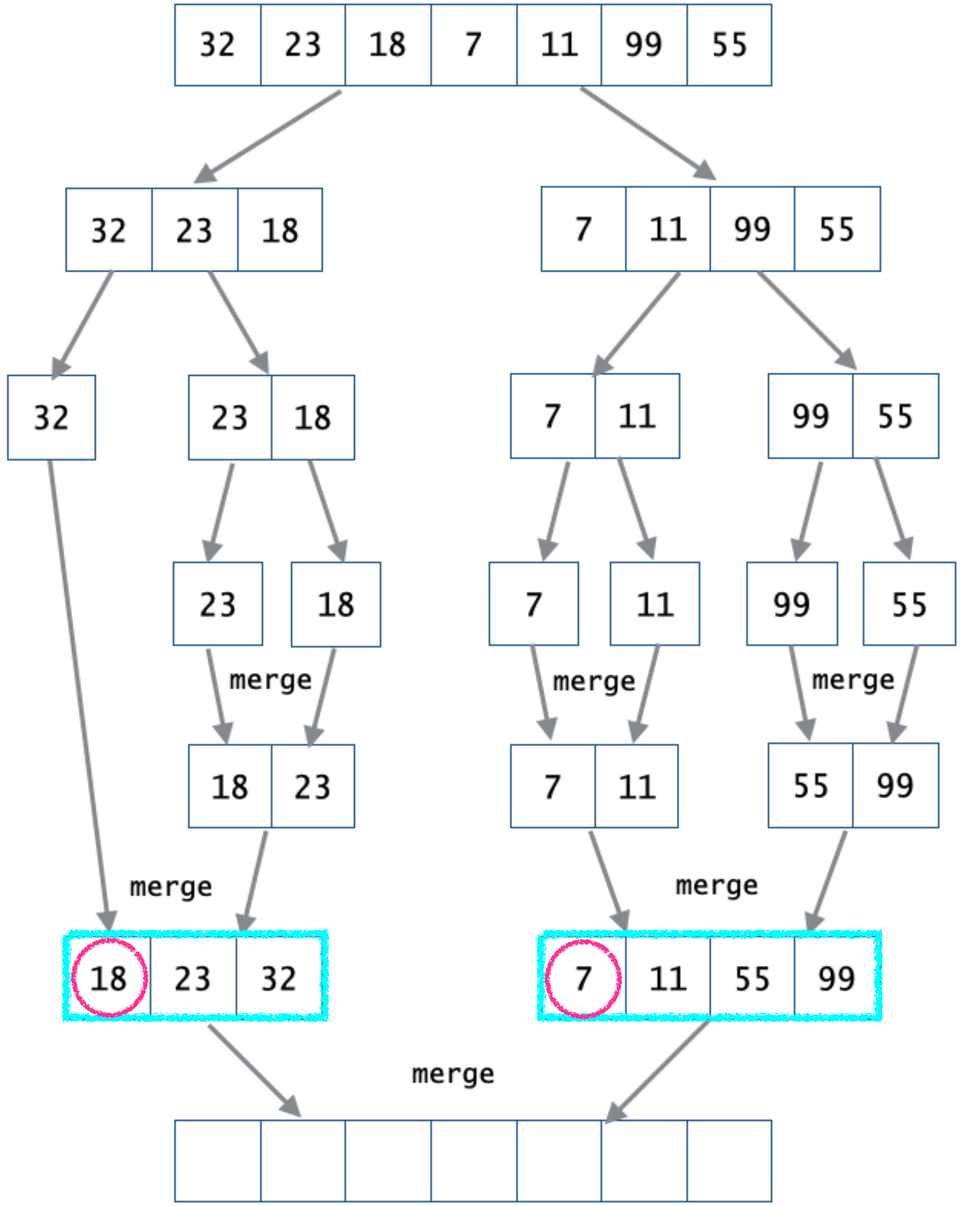
합병정렬

Merge Sort



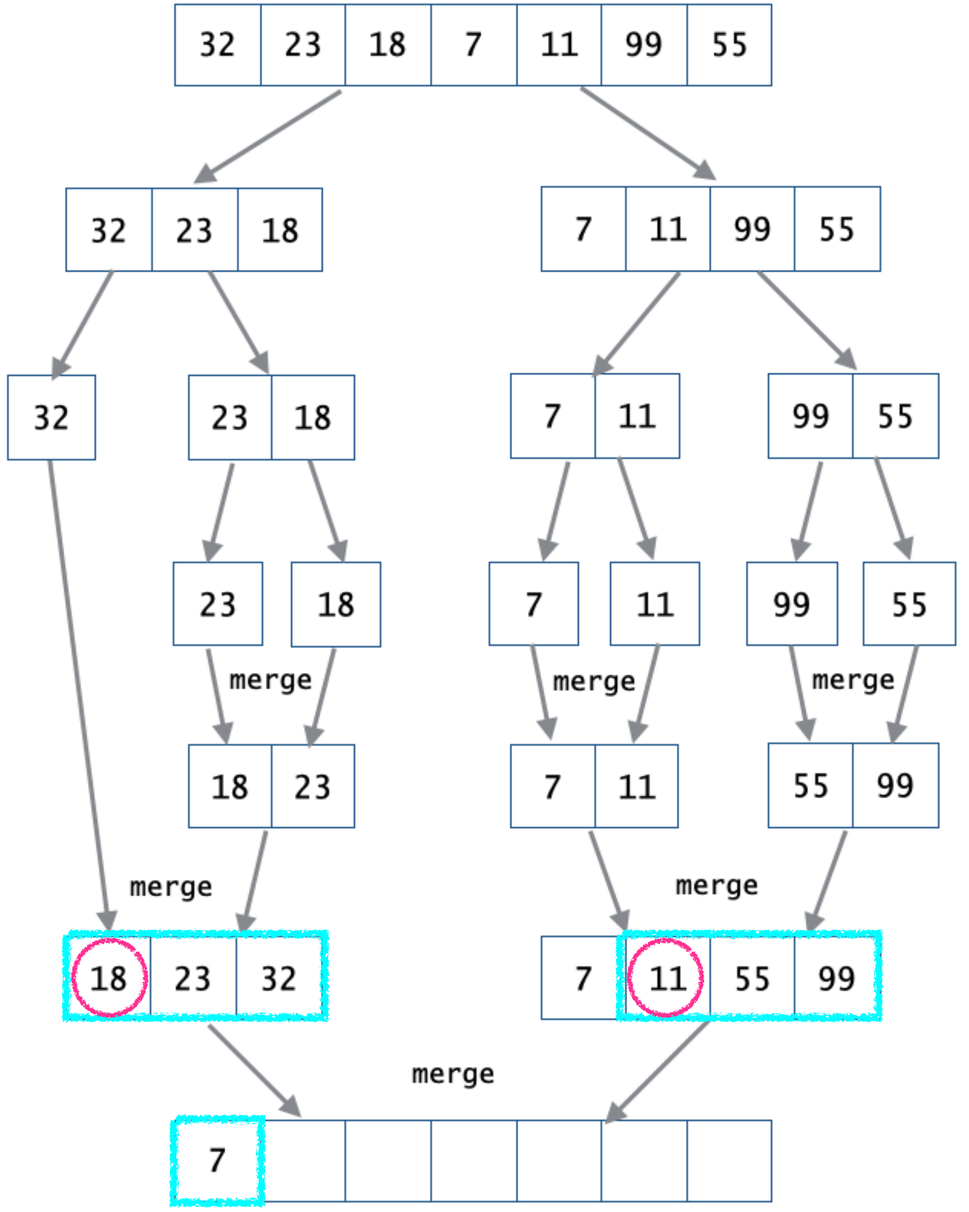
합병정렬

Merge Sort



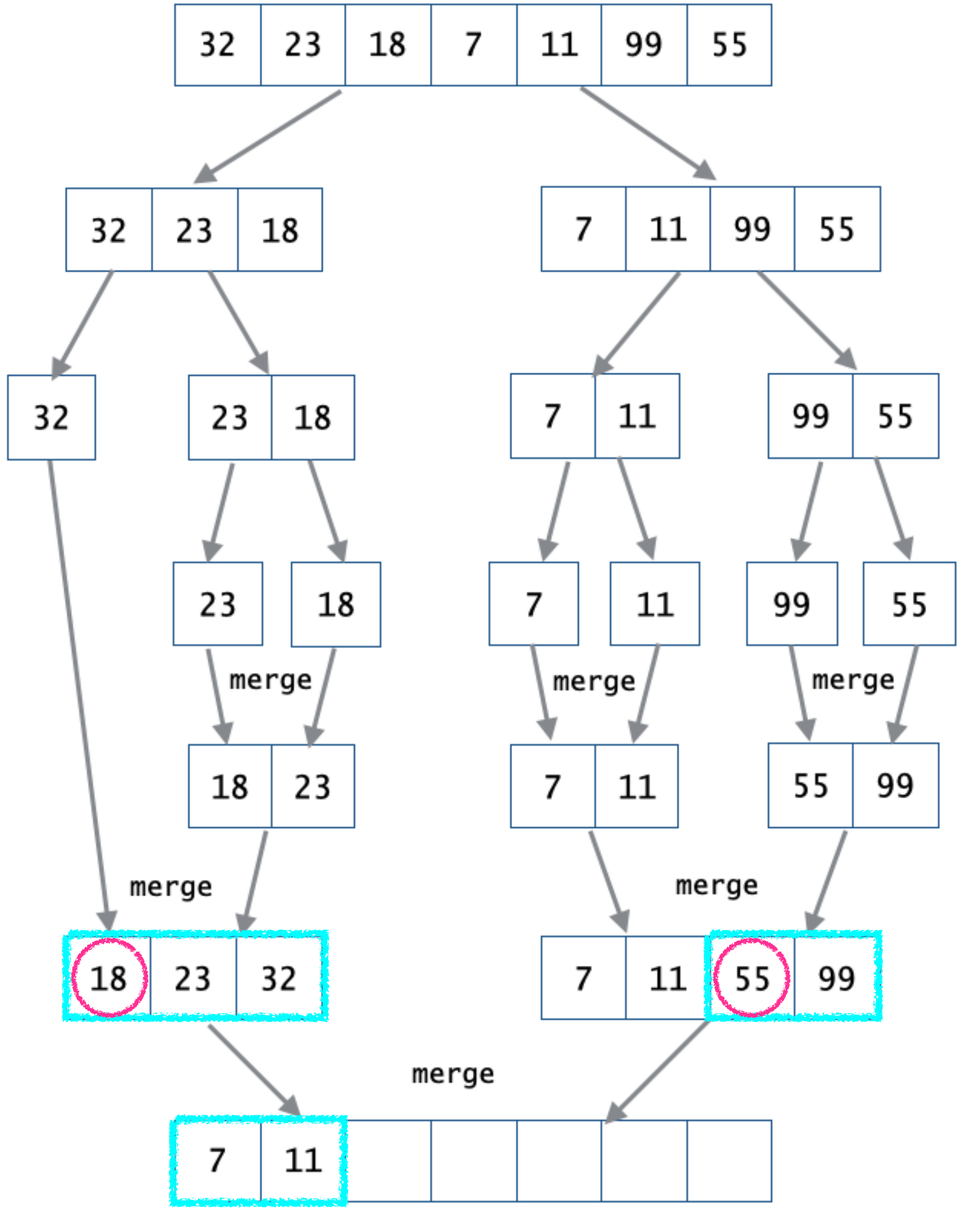
합병정렬

Merge Sort



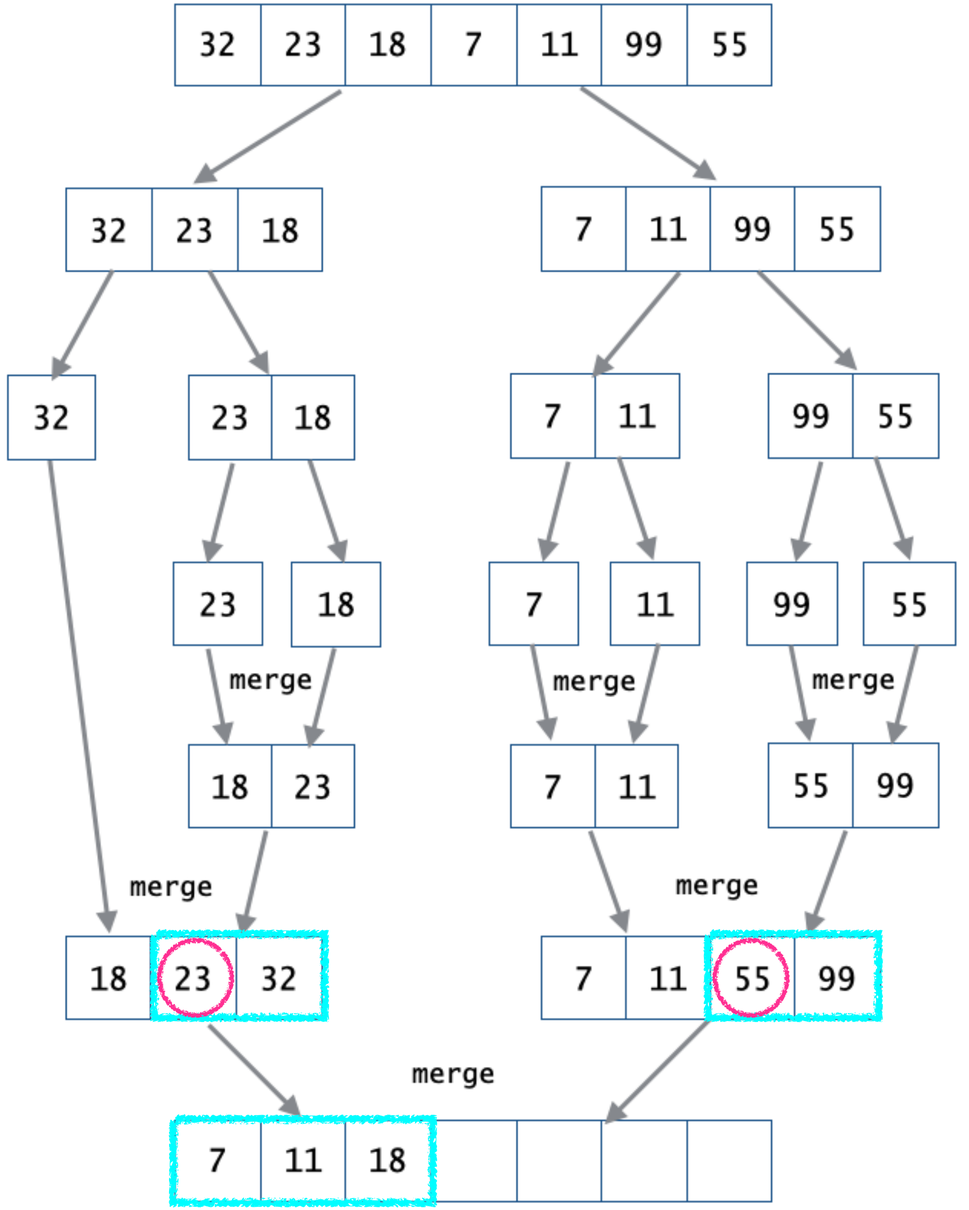
합병정렬

Merge Sort



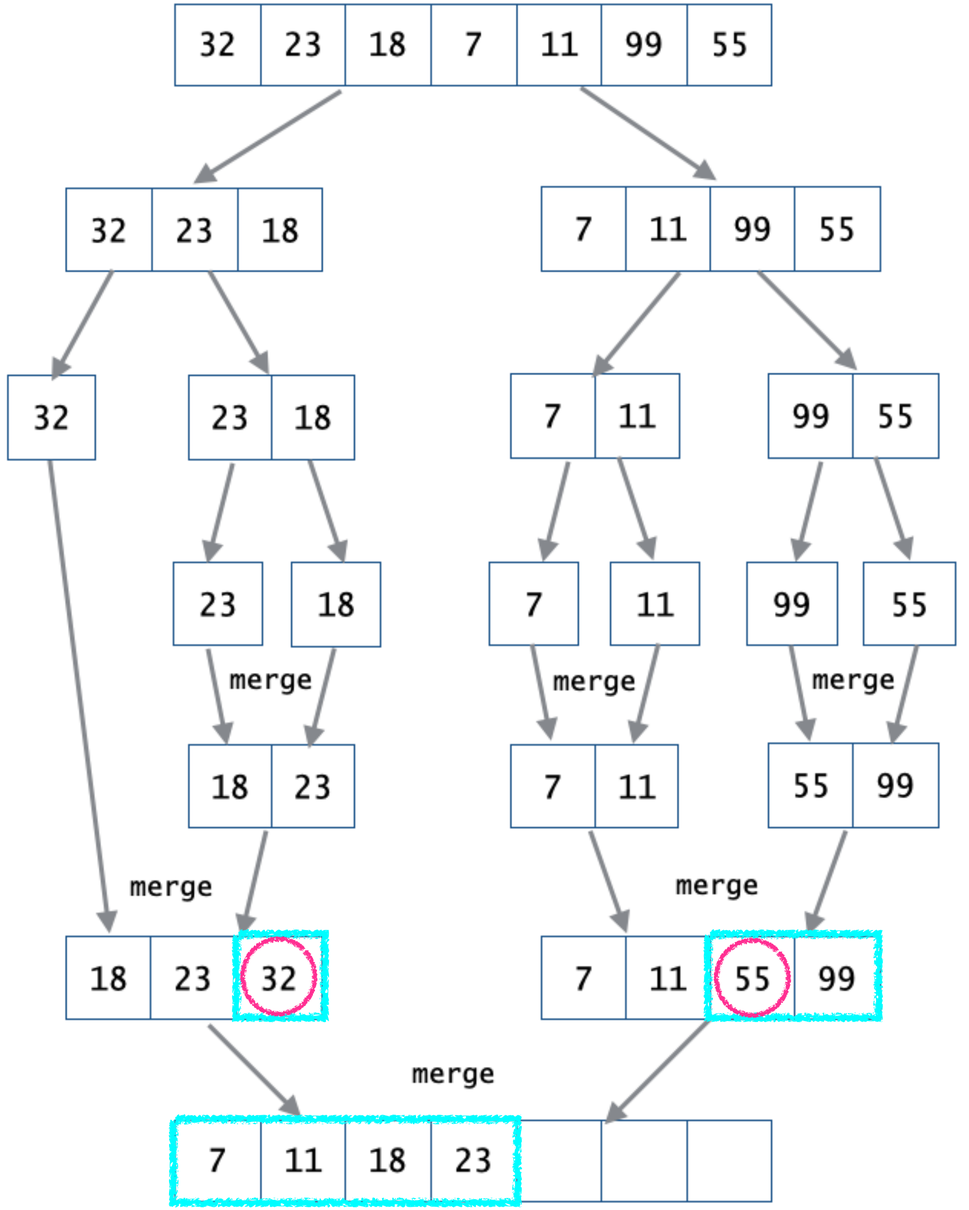
합병정렬

Merge Sort



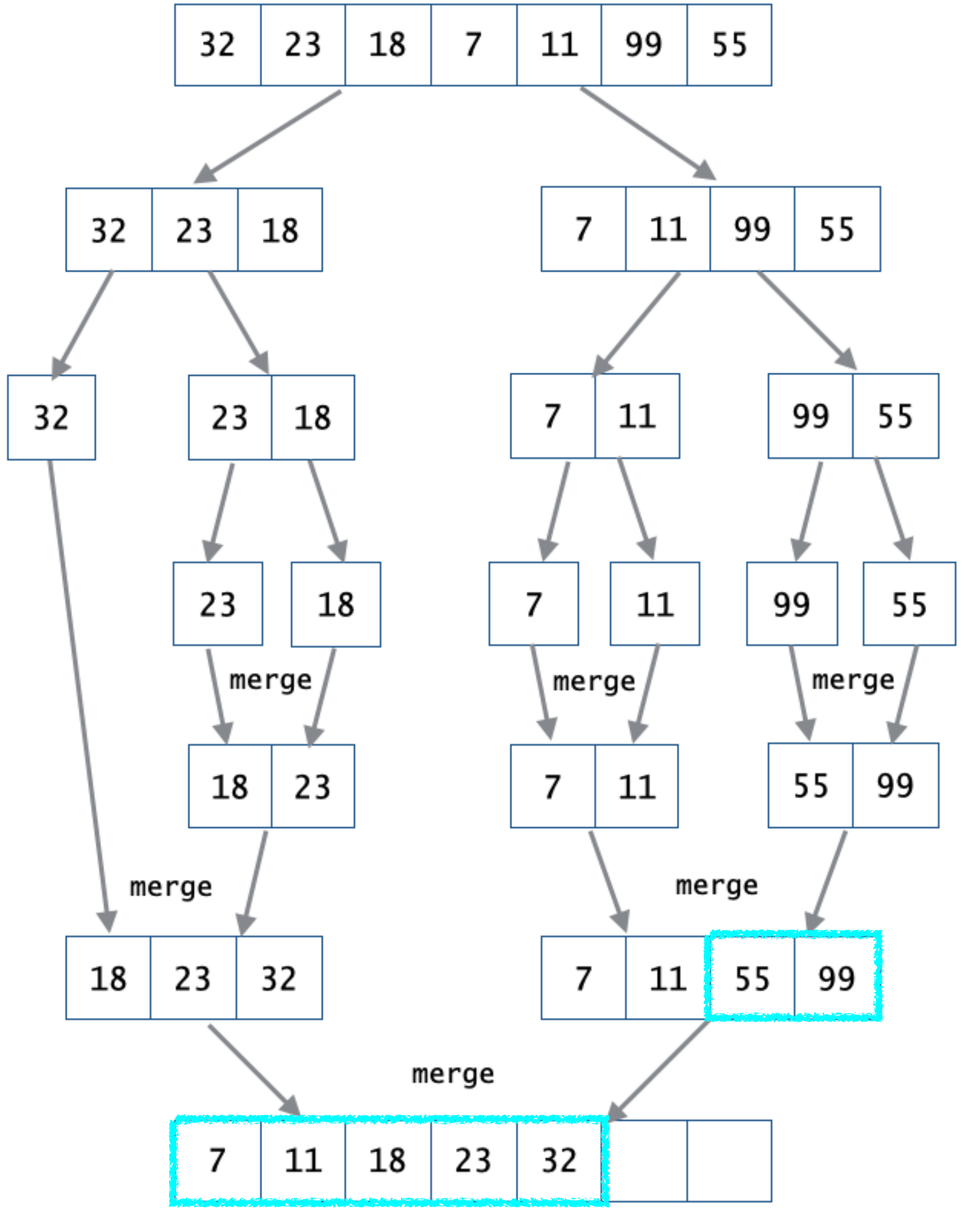
합병정렬

Merge Sort



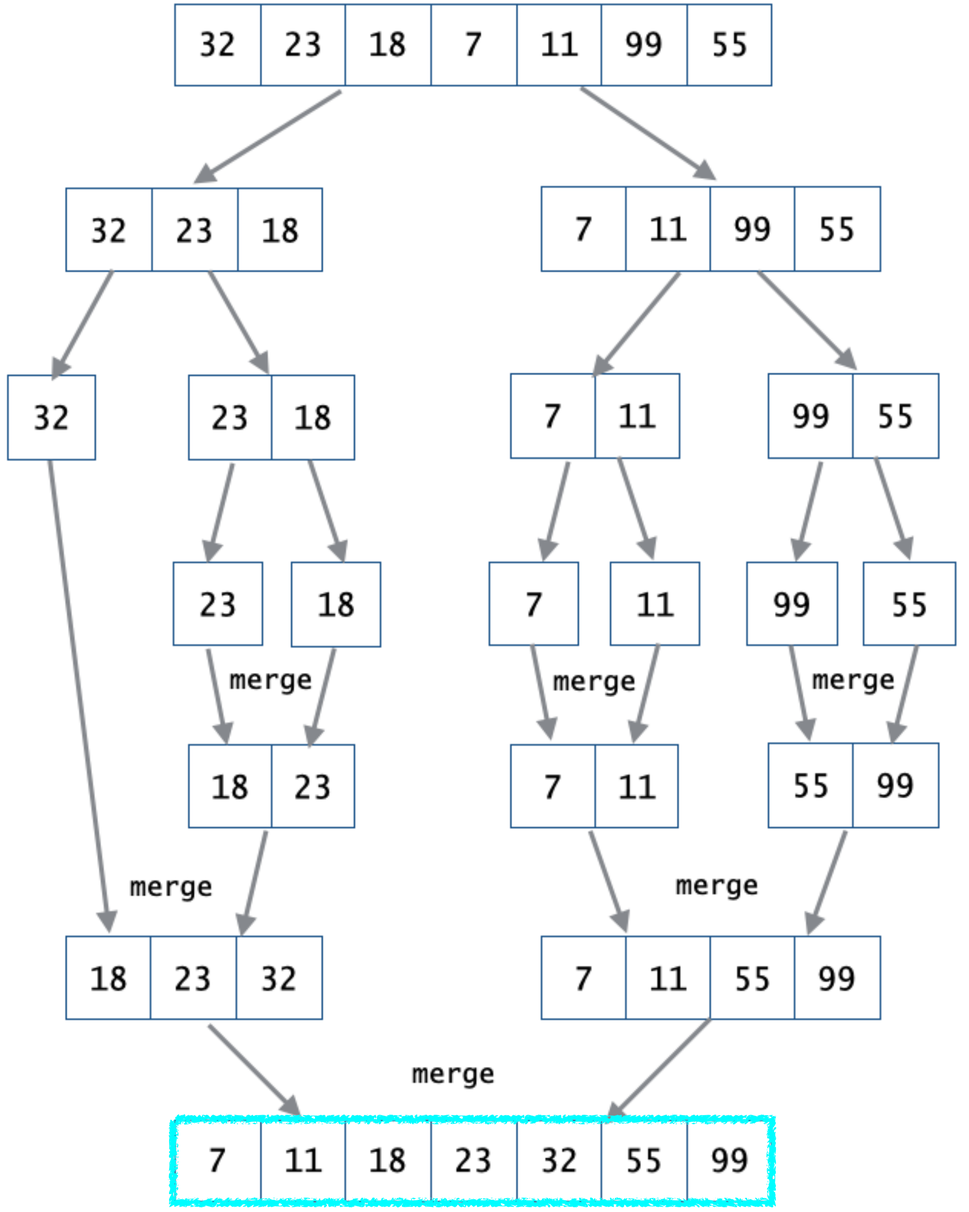
합병정렬

Merge Sort



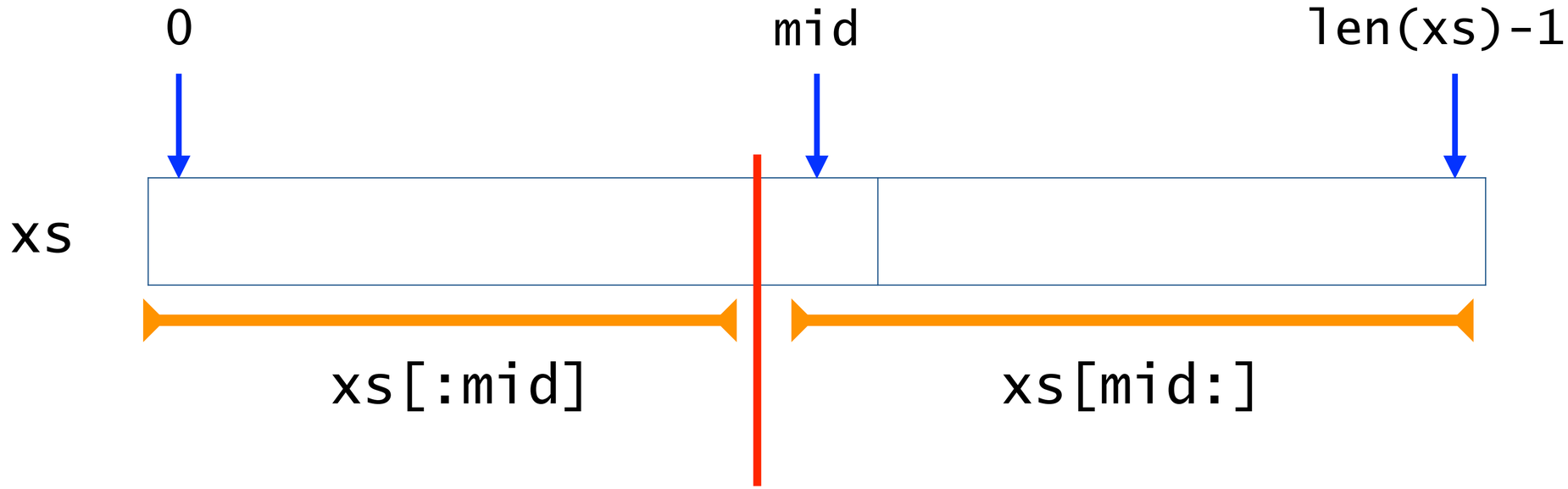
합병정렬

Merge Sort



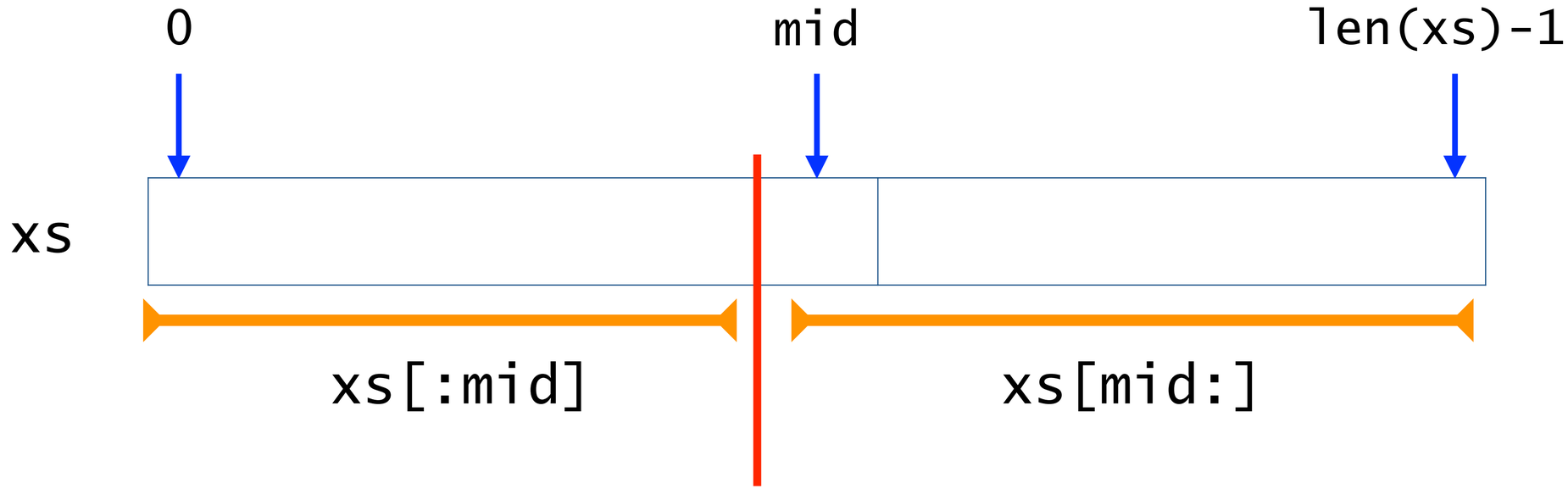
파이썬 리스트 반 나누기

```
mid = len(xs) // 2
```

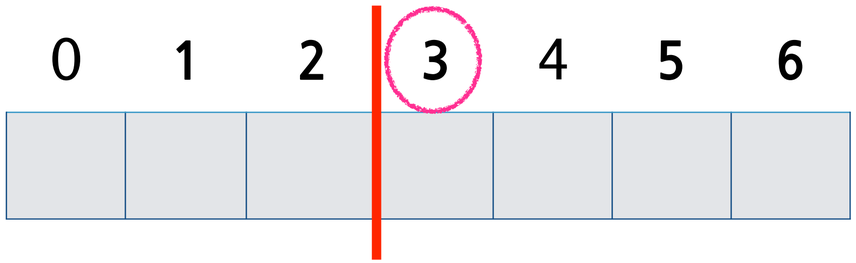


파이썬 리스트 반 나누기

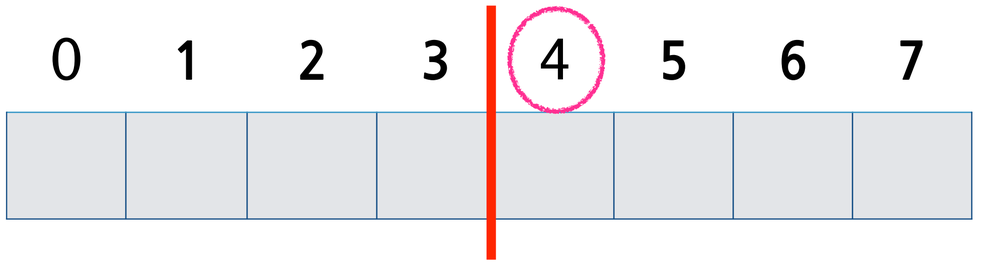
```
mid = len(xs) // 2
```




```
mid = 7 // 2 = 3
```




```
mid = 8 // 2 = 4
```



합병정렬

Merge Sort

code : 5-24.py

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

합병
Merge

정렬된 리스트 left와 right를 합병하는 재귀 함수 merge의 반복조건

```
  left와 right 양쪽에 원소가 최소한 하나씩은 있음
= left != [] and right != []
= not (left == []) and not (right == [])
= not (left == [] or right == [])
```

합병
Merge

code : 5-25.py

```
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
```

pp.242~243

실습 5.8 merge : 꼬리재귀 함수 버전

위의 재귀함수 merge는 꼬리재귀가 아니다. append 메소드를 활용하여 꼬리재귀 함수를 다음 뼈대코드에 맞추어 완성하자.

code : 5-26.py

```
1  def merge(left,right):
2      def loop(left,right,ss):
3          if not (left == [] or right == []):
4              if left[0] <= right[0]:
5                  pass # Write your code here.
6
7              else:
8                  pass # Write your code here.
9
10         else:
11             return ss + left + right
12     return loop(left,right,[])
```

실습 5.9 merge : while 루프 버전

〈실습 5.8〉에서 작성한 꼬리재귀 함수를 참고하여, merge 함수의 while 루프 버전을 다음 뼈대코드에 맞추어 완성하자.

code : 5-27.py

```
1  def merge(left,right):
2      ss = []
3      while not (left == [] or right == []):
4          if left[0] <= right[0]:
5              pass # Write your code here.
6
7          else:
8              pass # Write your code here.
9
10     return ss + left + right
```

퀵정렬

Quicksort

Tony Hoare

퀵정렬 알고리즘
Quicksort

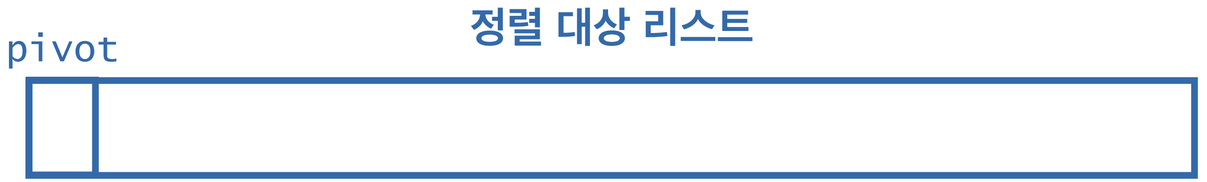
리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • `pivot`을 기준으로 작은 원소는 왼쪽 리스트 `left`로, 큰 원소는 오른쪽 리스트 `right`로 옮긴다. • 왼쪽 리스트 `left`와 오른쪽 리스트 `right`를 각각 재귀로 정렬하고, `left`와 `pivot`과 `right`를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • `pivot`을 기준으로 작은 원소는 왼쪽 리스트 `left`로, 큰 원소는 오른쪽 리스트 `right`로 옮긴다. • 왼쪽 리스트 `left`와 오른쪽 리스트 `right`를 각각 재귀로 정렬하고, `left`와 `pivot`과 `right`를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

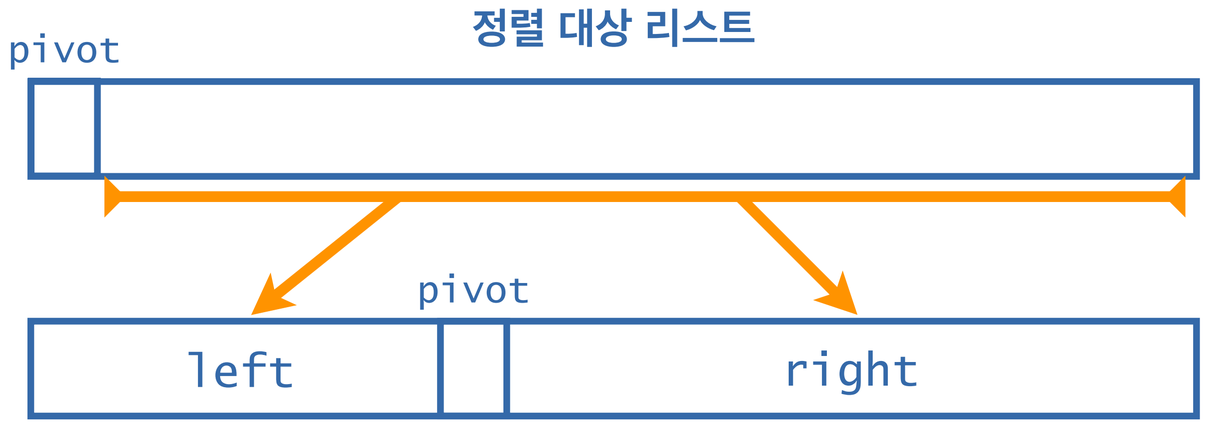
리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • `pivot`을 기준으로 작은 원소는 왼쪽 리스트 `left`로, 큰 원소는 오른쪽 리스트 `right`로 옮긴다. • 왼쪽 리스트 `left`와 오른쪽 리스트 `right`를 각각 재귀로 정렬하고, `left`와 `pivot`과 `right`를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

정렬 대상 리스트

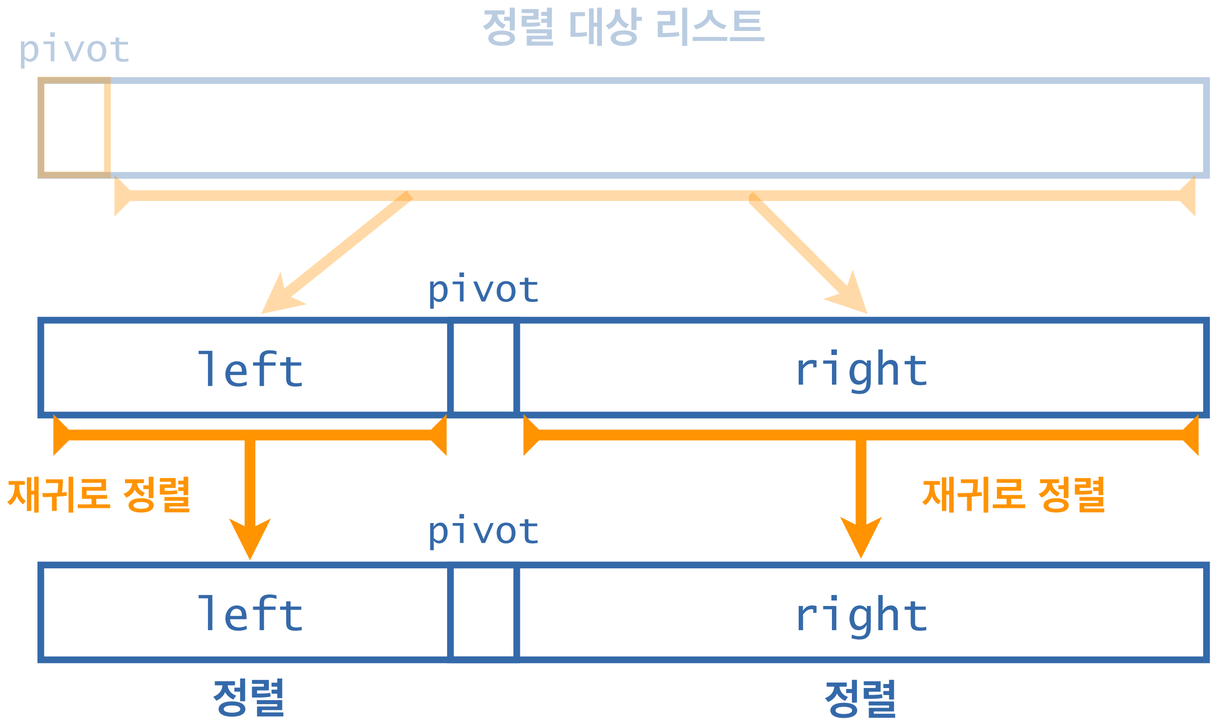
리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • `pivot`을 기준으로 작은 원소는 왼쪽 리스트 `left`로, 큰 원소는 오른쪽 리스트 `right`로 옮긴다. • 왼쪽 리스트 `left`와 오른쪽 리스트 `right`를 각각 재귀로 정렬하고, `left`와 `pivot`과 `right`를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.



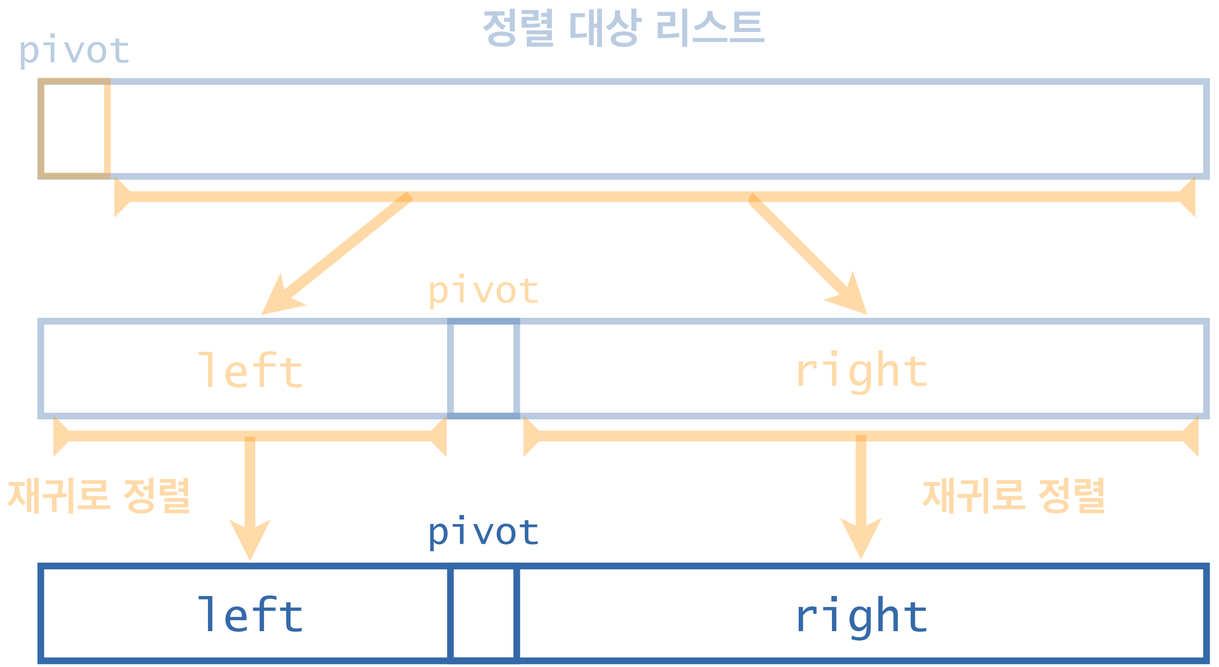
리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 pivot을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • pivot을 기준으로 작은 원소는 왼쪽 리스트 left로, 큰 원소는 오른쪽 리스트 right로 옮긴다. • 왼쪽 리스트 left와 오른쪽 리스트 right를 각각 재귀로 정렬하고, left와 pivot과 right를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.



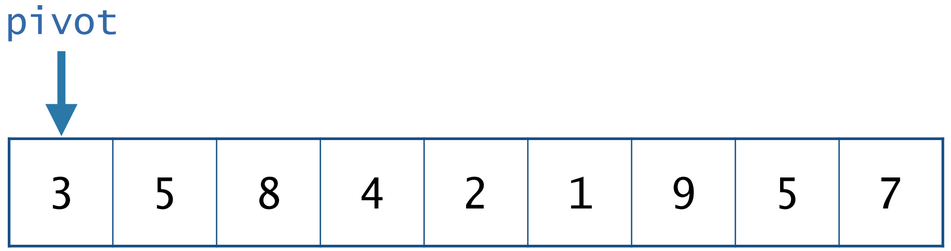
리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 pivot을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • pivot을 기준으로 작은 원소는 왼쪽 리스트 left로, 큰 원소는 오른쪽 리스트 right로 옮긴다. • 왼쪽 리스트 left와 오른쪽 리스트 right를 각각 재귀로 정렬하고, left와 pivot과 right를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.



리스트 xs를 퀵정렬하려면		
반복 조건	`len(xs) > 1`	• 기준으로 사용할 피봇 원소 `pivot`을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • `pivot`을 기준으로 작은 원소는 왼쪽 리스트 `left`로, 큰 원소는 오른쪽 리스트 `right`로 옮긴다. • 왼쪽 리스트 `left`와 오른쪽 리스트 `right`를 각각 재귀로 정렬하고, `left`와 `pivot`과 `right`를 나란히 붙여서 리턴한다.
종료 조건	`len(xs) <= 1`	• 정렬할 필요가 없으므로 그대로 리턴한다.

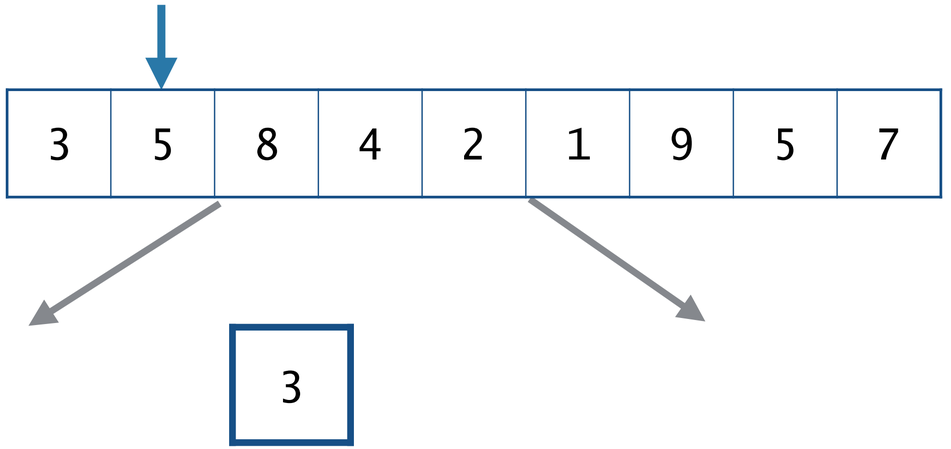


정렬



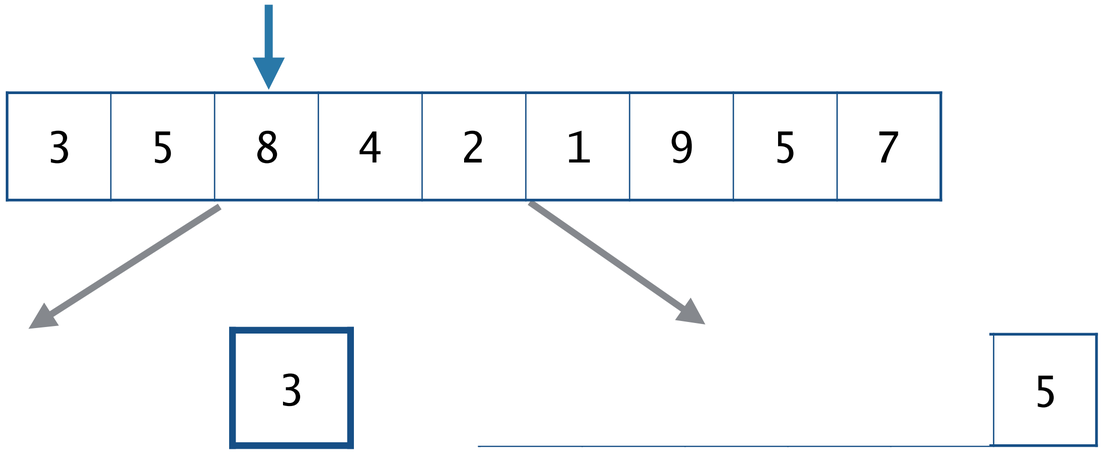
퀵정렬

Quicksort



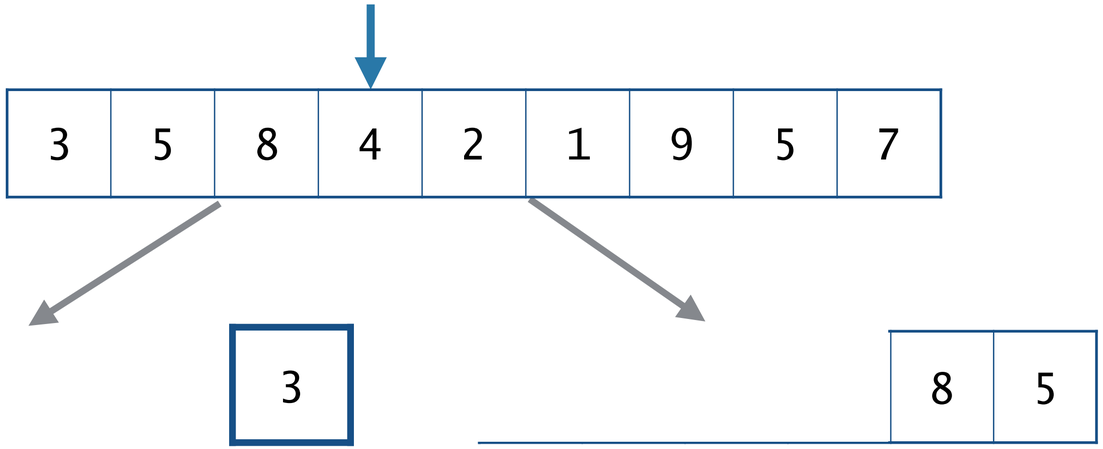
퀵정렬

Quicksort



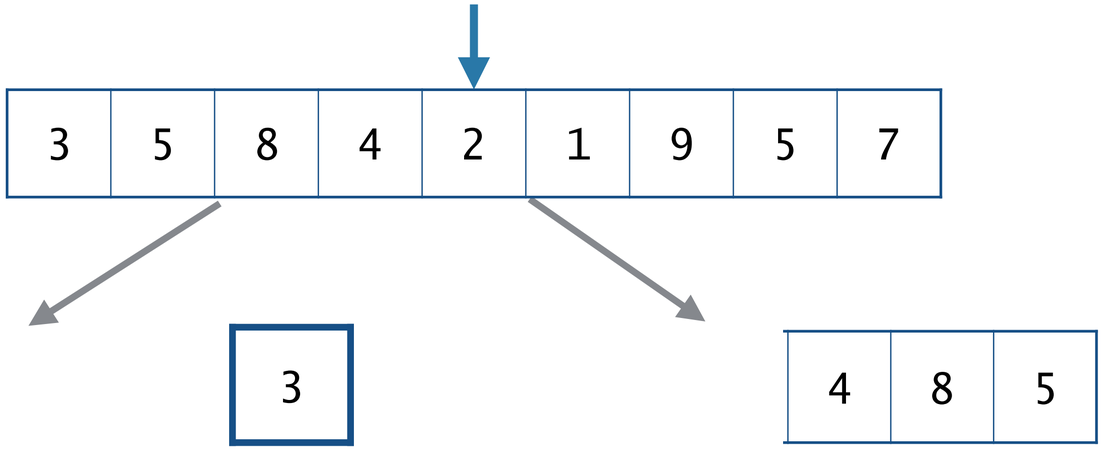
퀵정렬

Quicksort



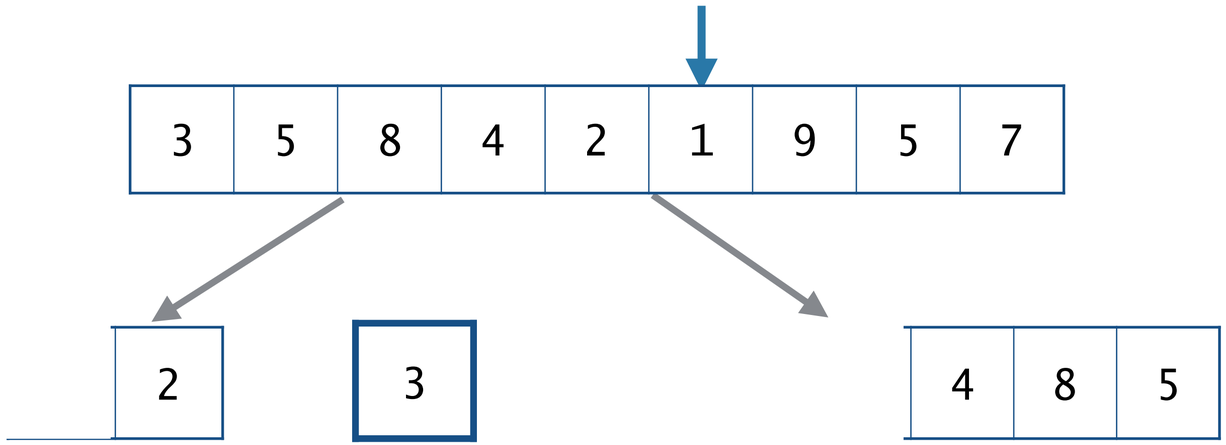
퀵정렬

Quicksort



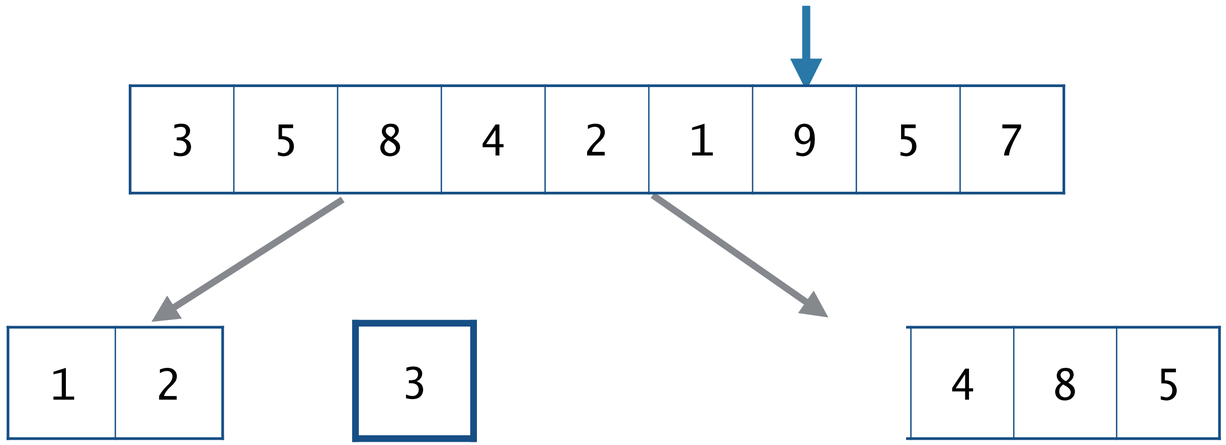
퀵정렬

Quicksort



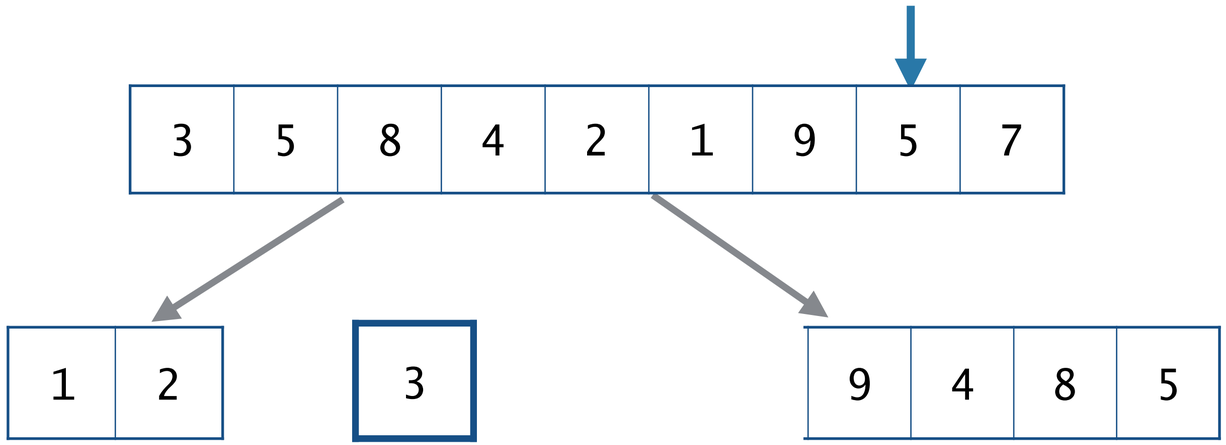
퀵정렬

Quicksort



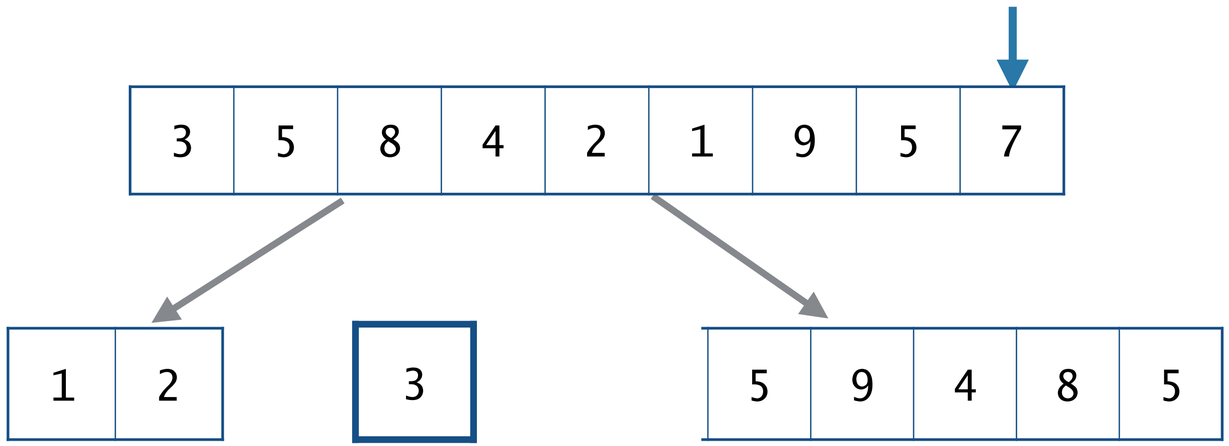
퀵정렬

Quicksort



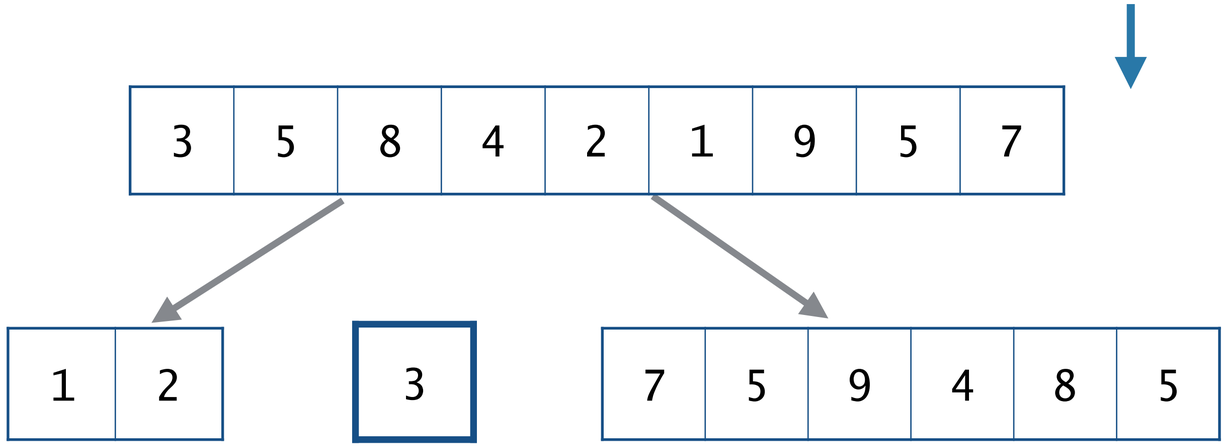
퀵정렬

Quicksort



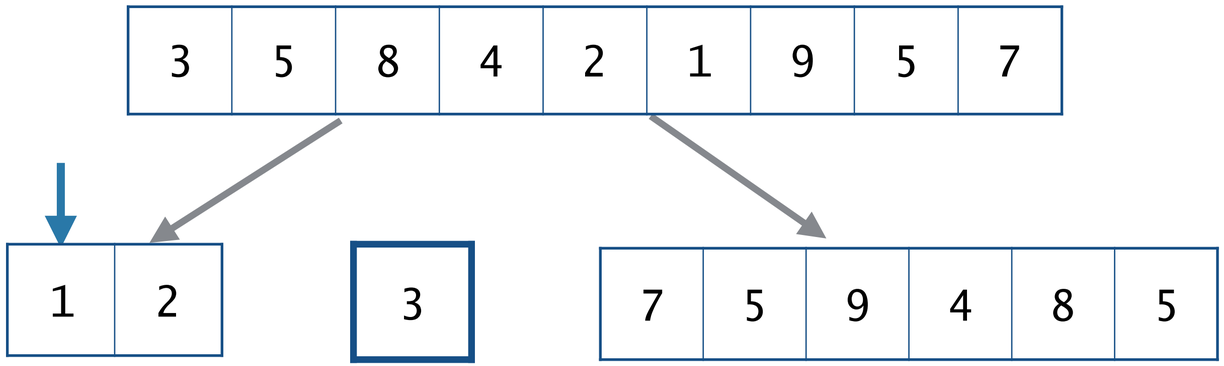
퀵정렬

Quicksort



퀵정렬

Quicksort

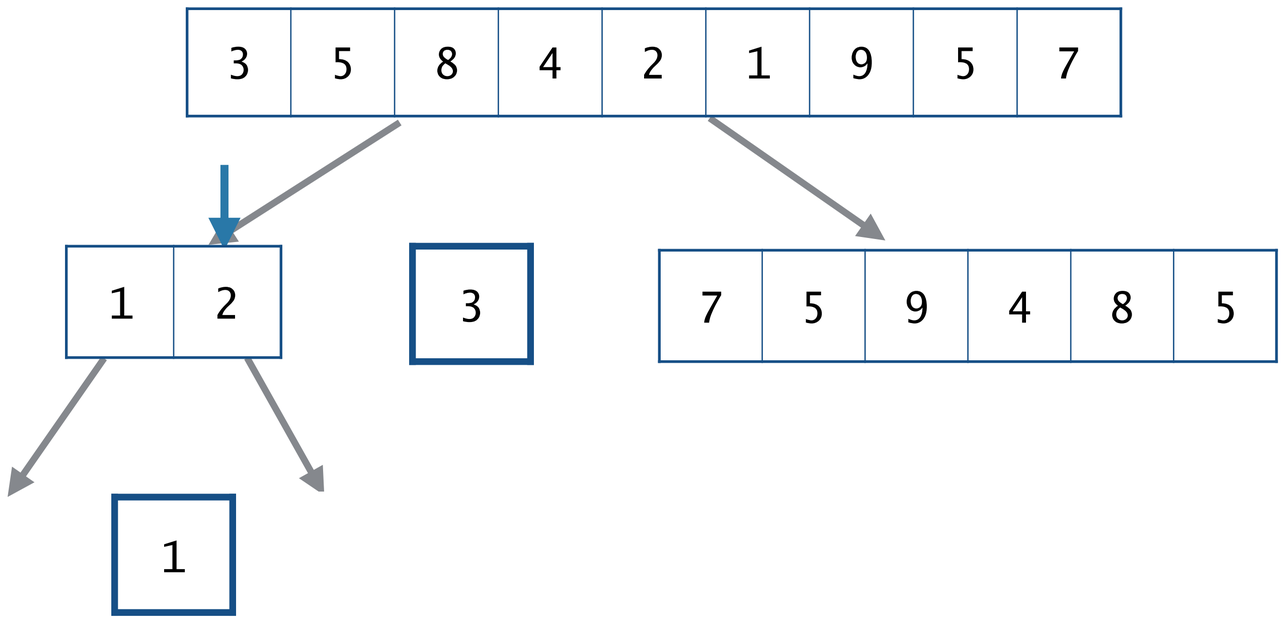


퀵정렬

Quicksort

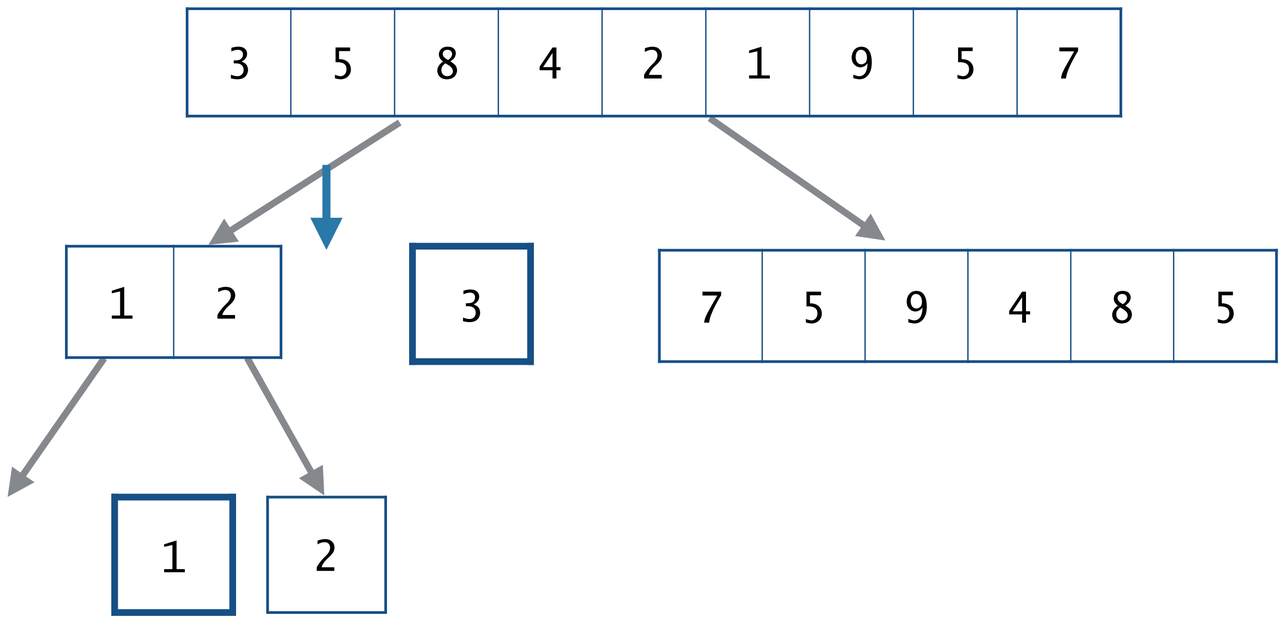
퀵정렬

Quicksort



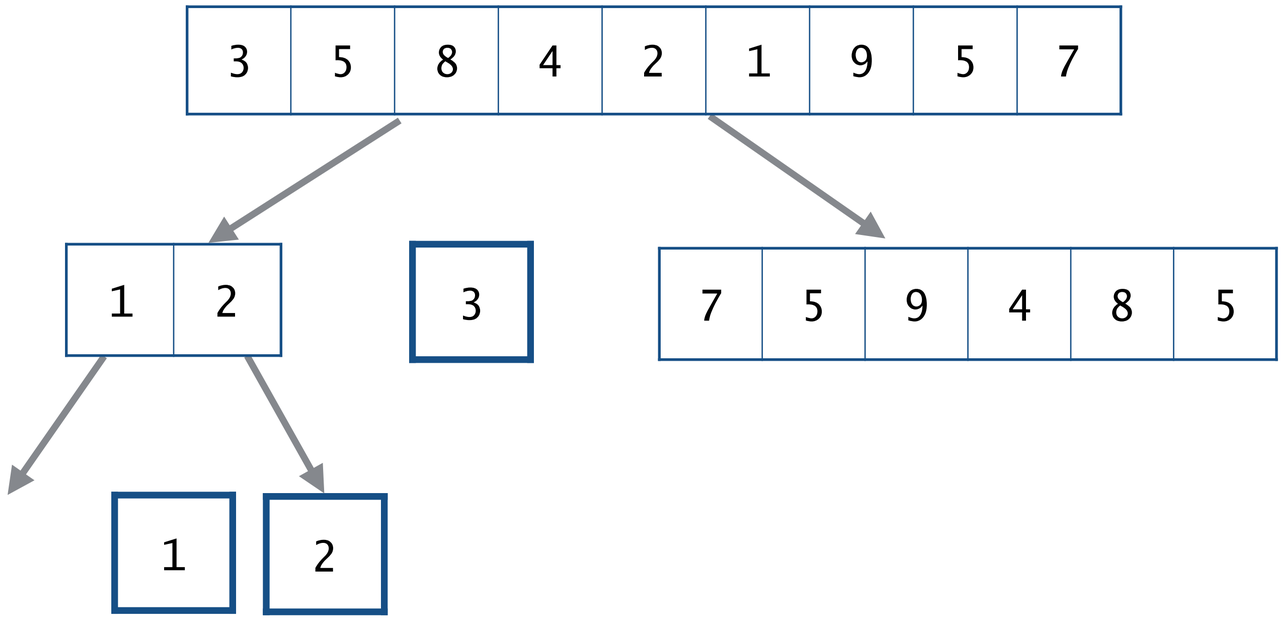
퀵정렬

Quicksort



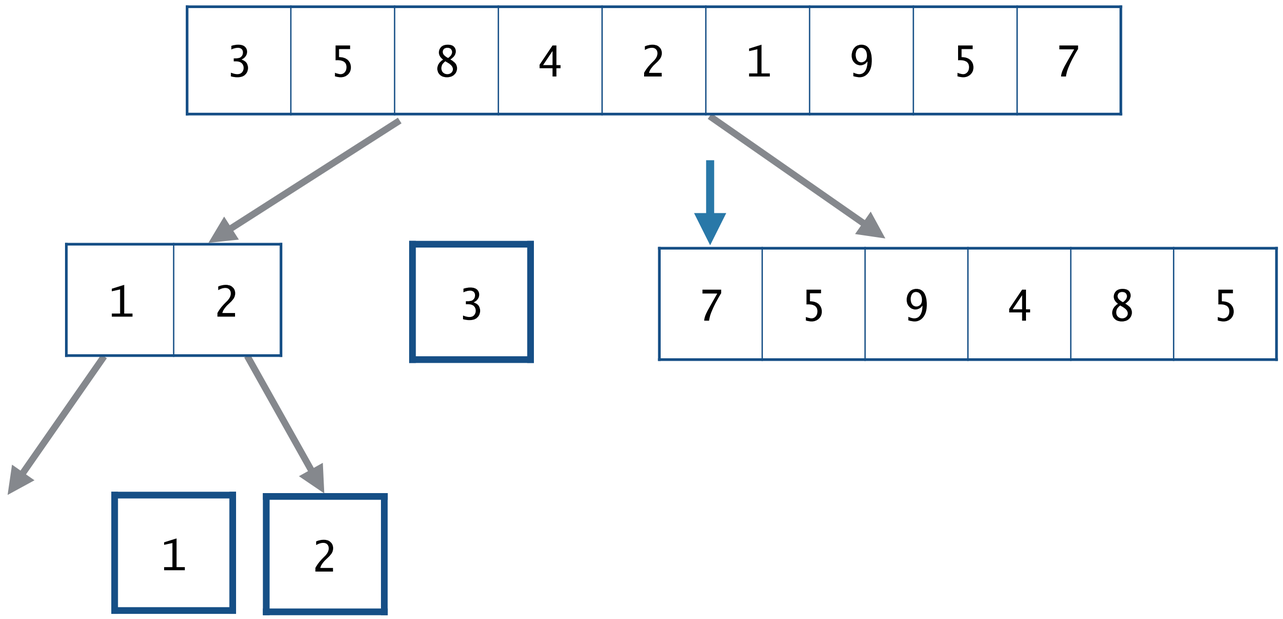
퀵정렬

Quicksort



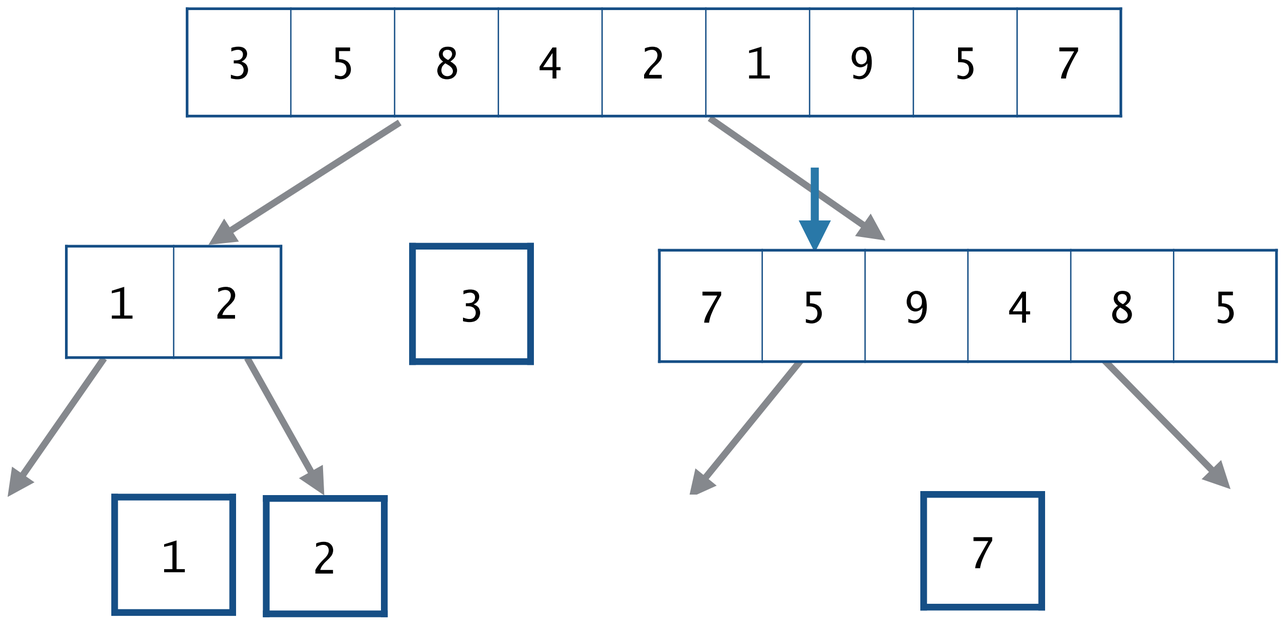
퀵정렬

Quicksort



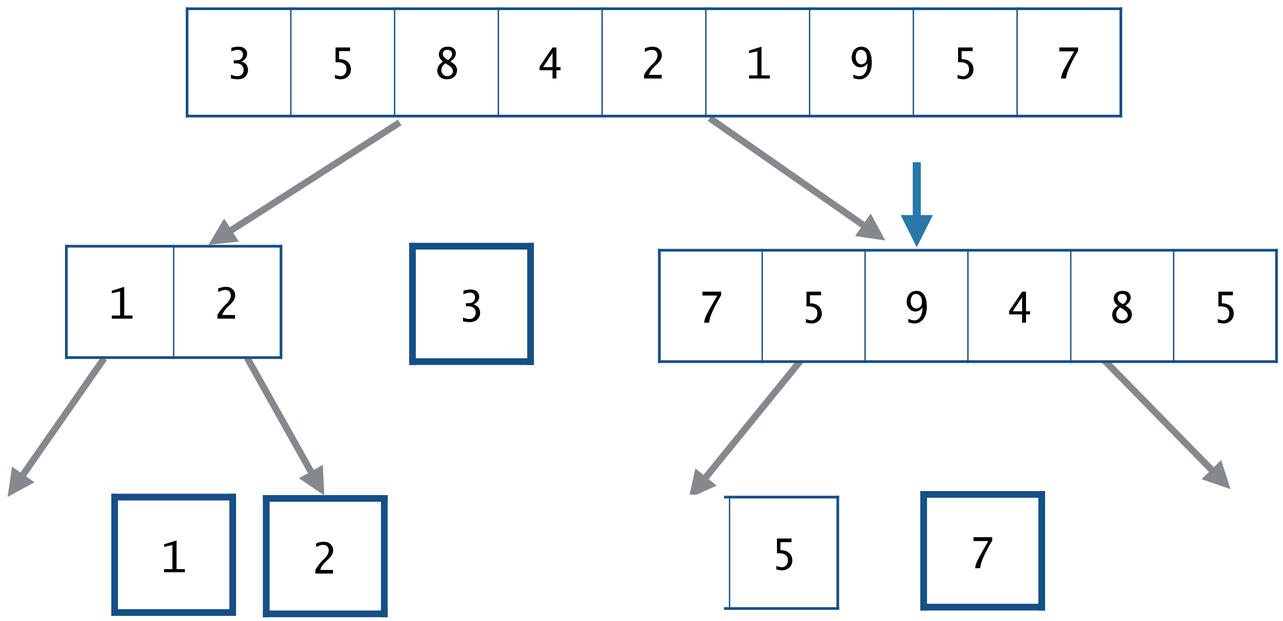
퀵정렬

Quicksort



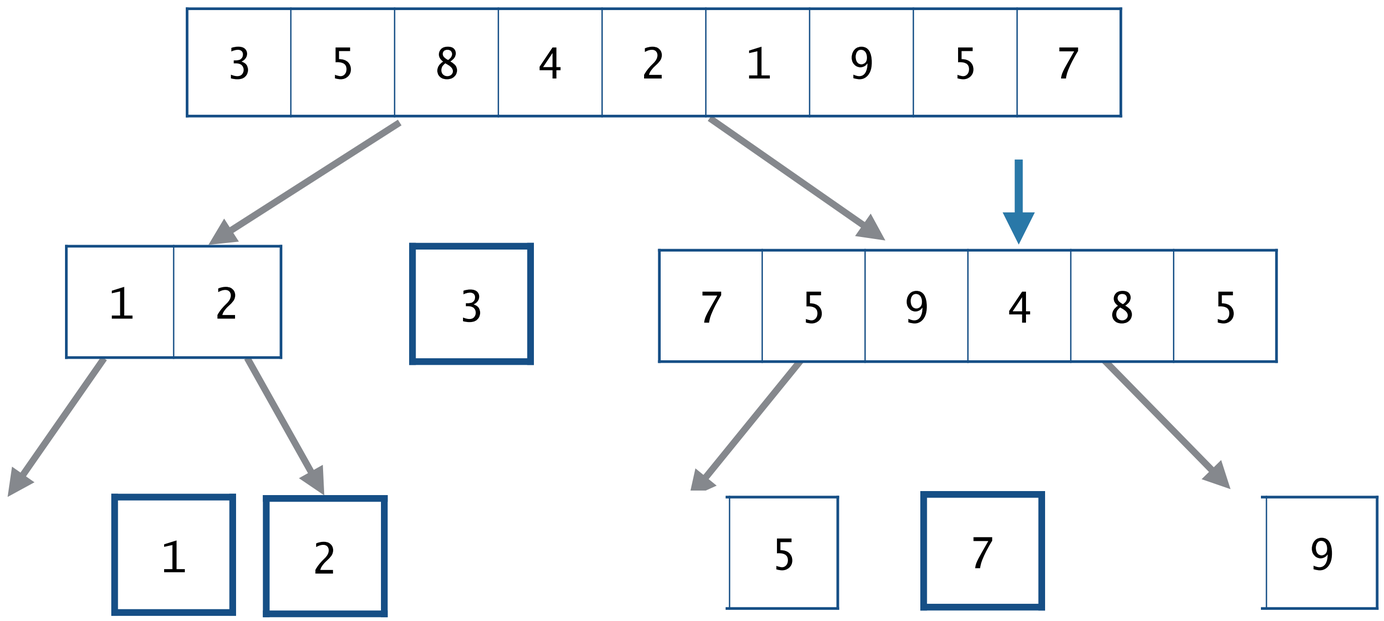
퀵정렬

Quicksort



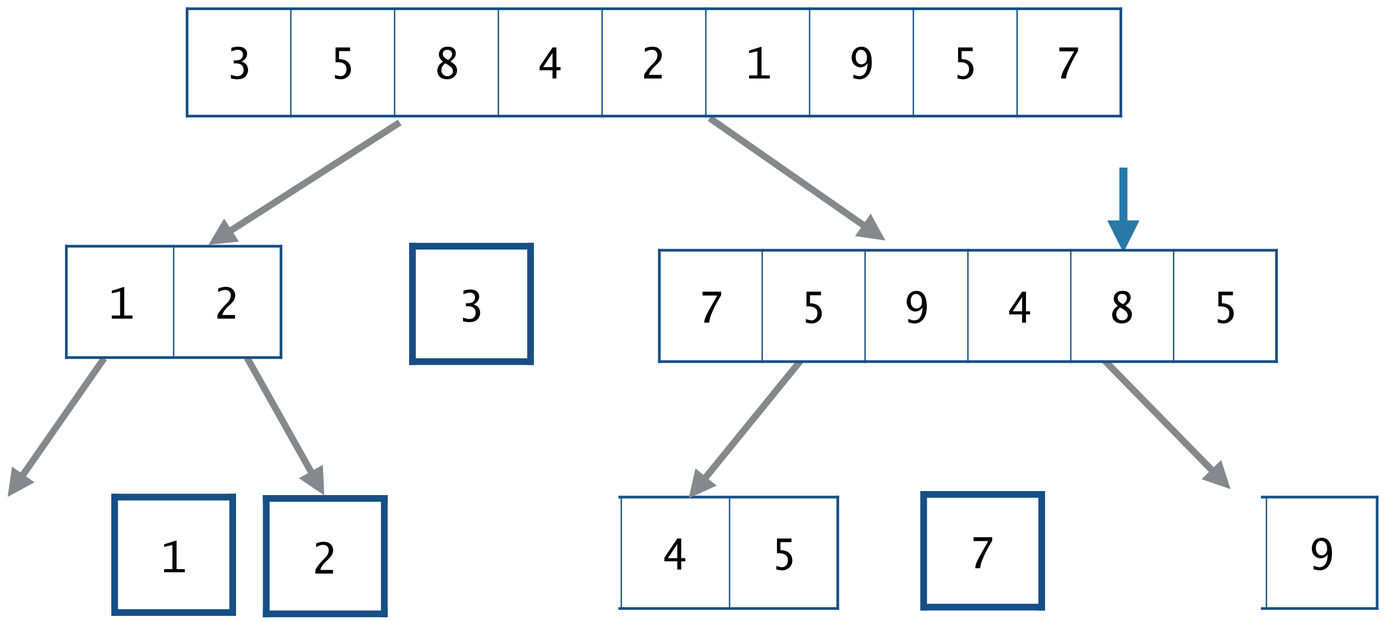
퀵정렬

Quicksort



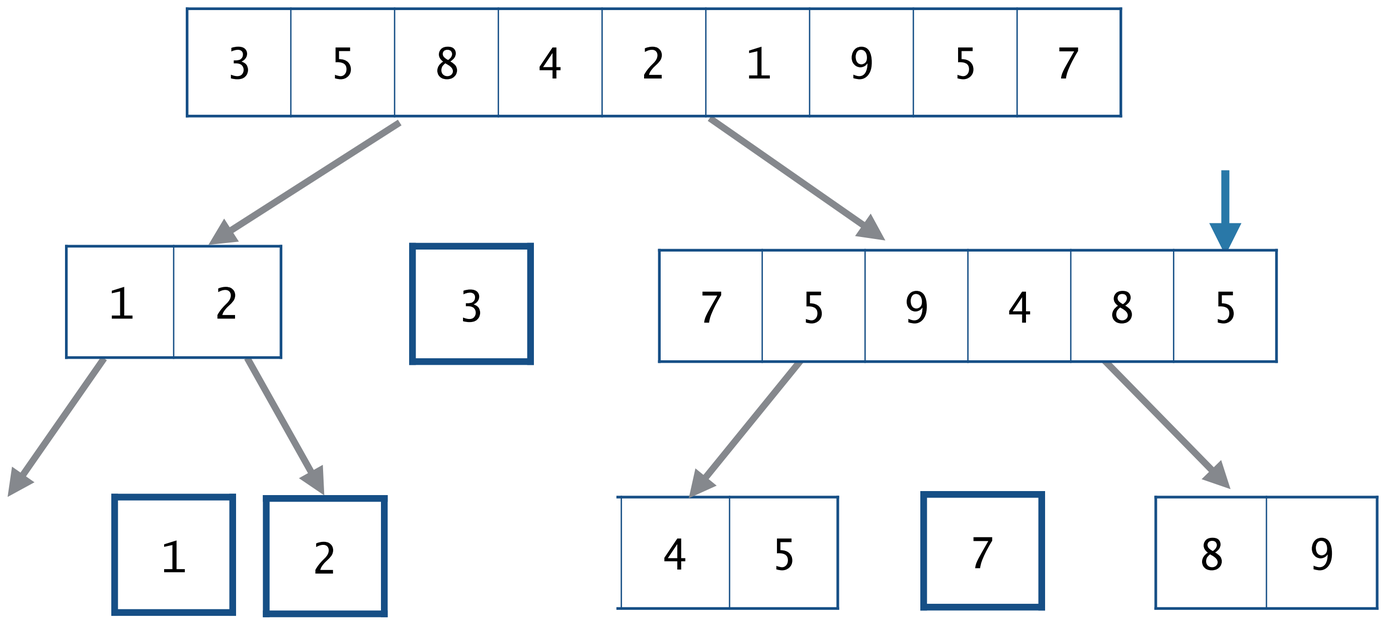
퀵정렬

Quicksort



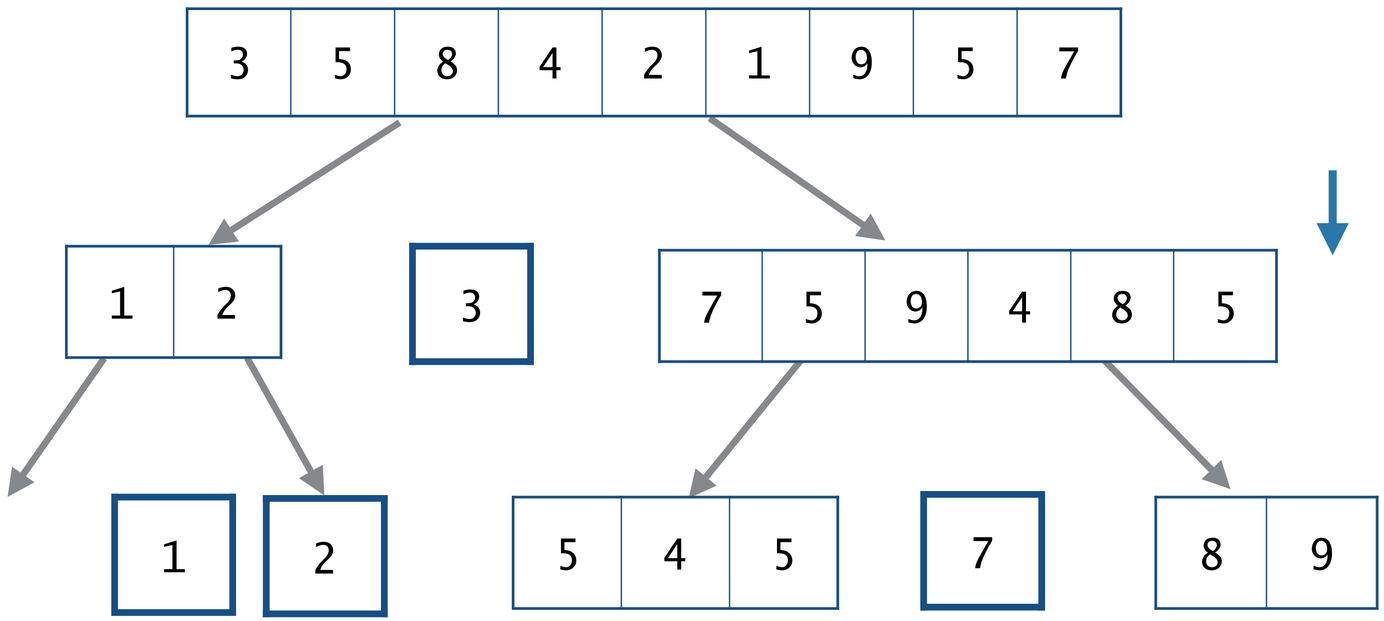
퀵정렬

Quicksort



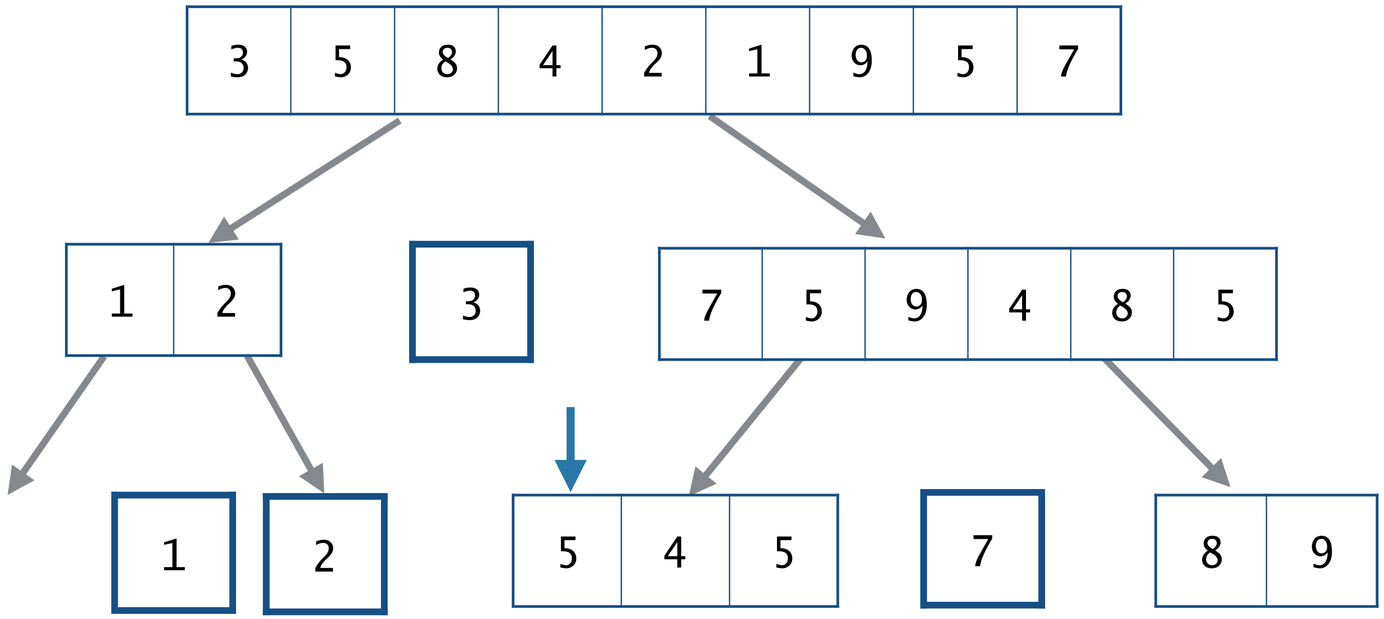
퀵정렬

Quicksort



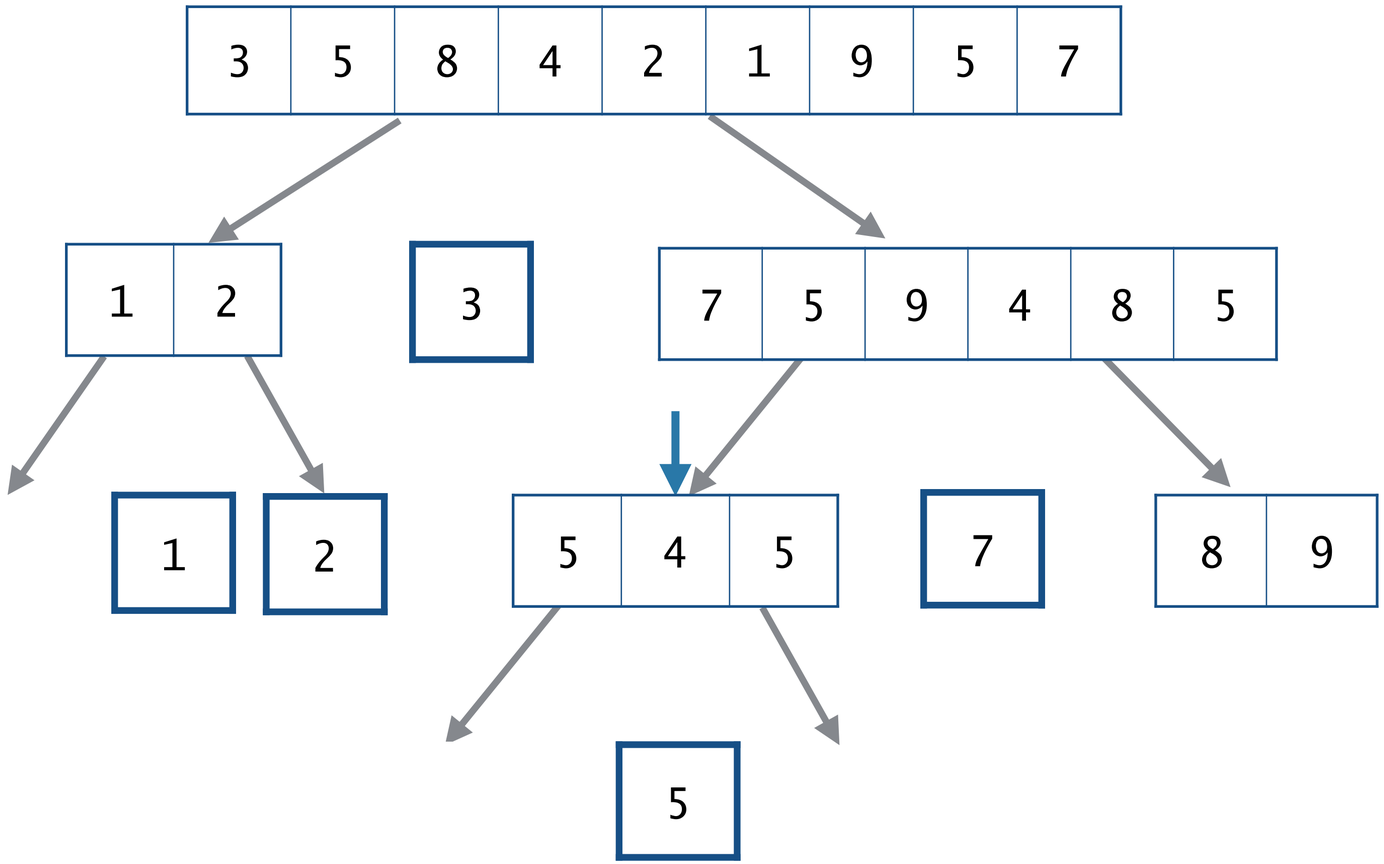
퀵정렬

Quicksort



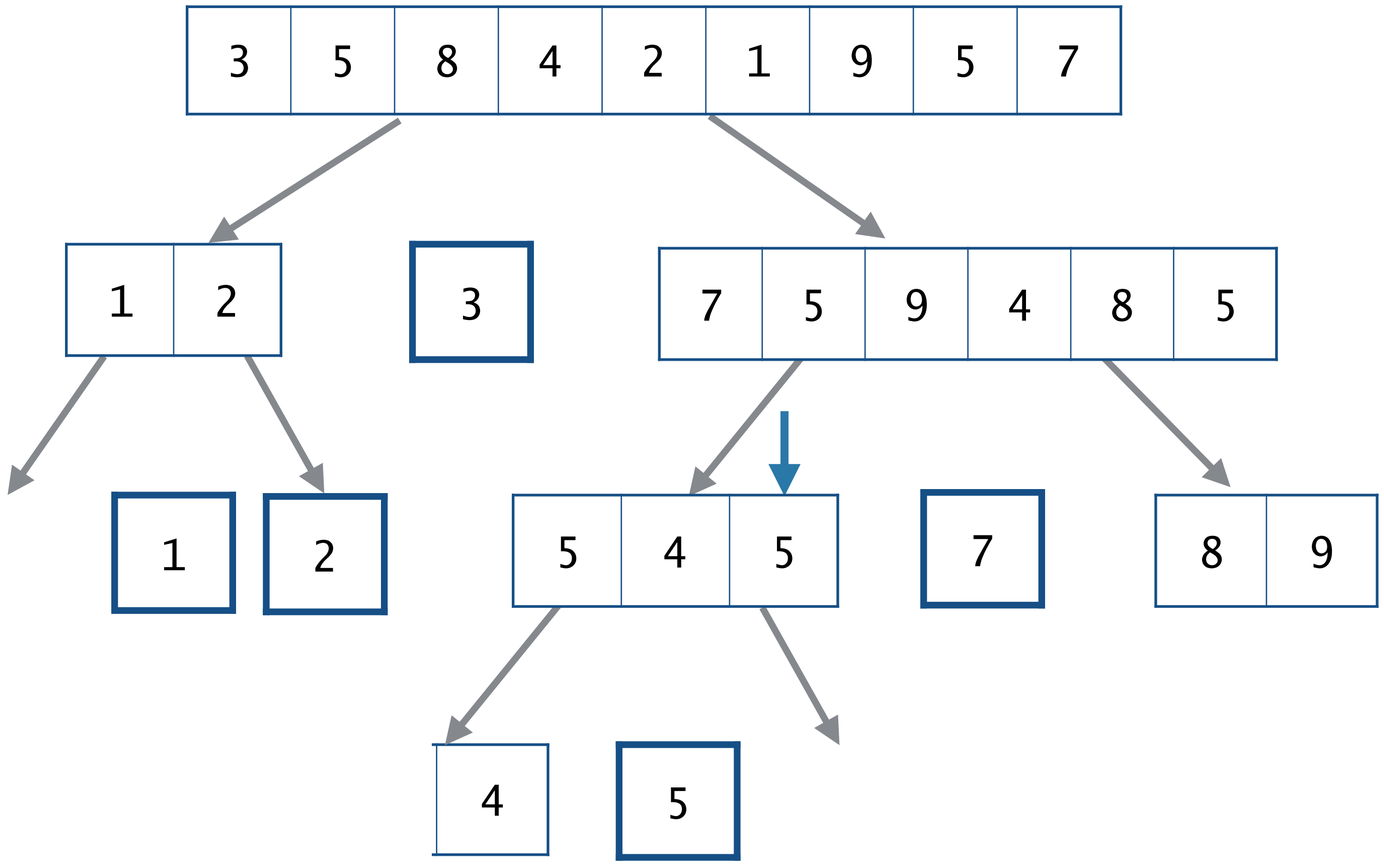
퀵정렬

Quicksort



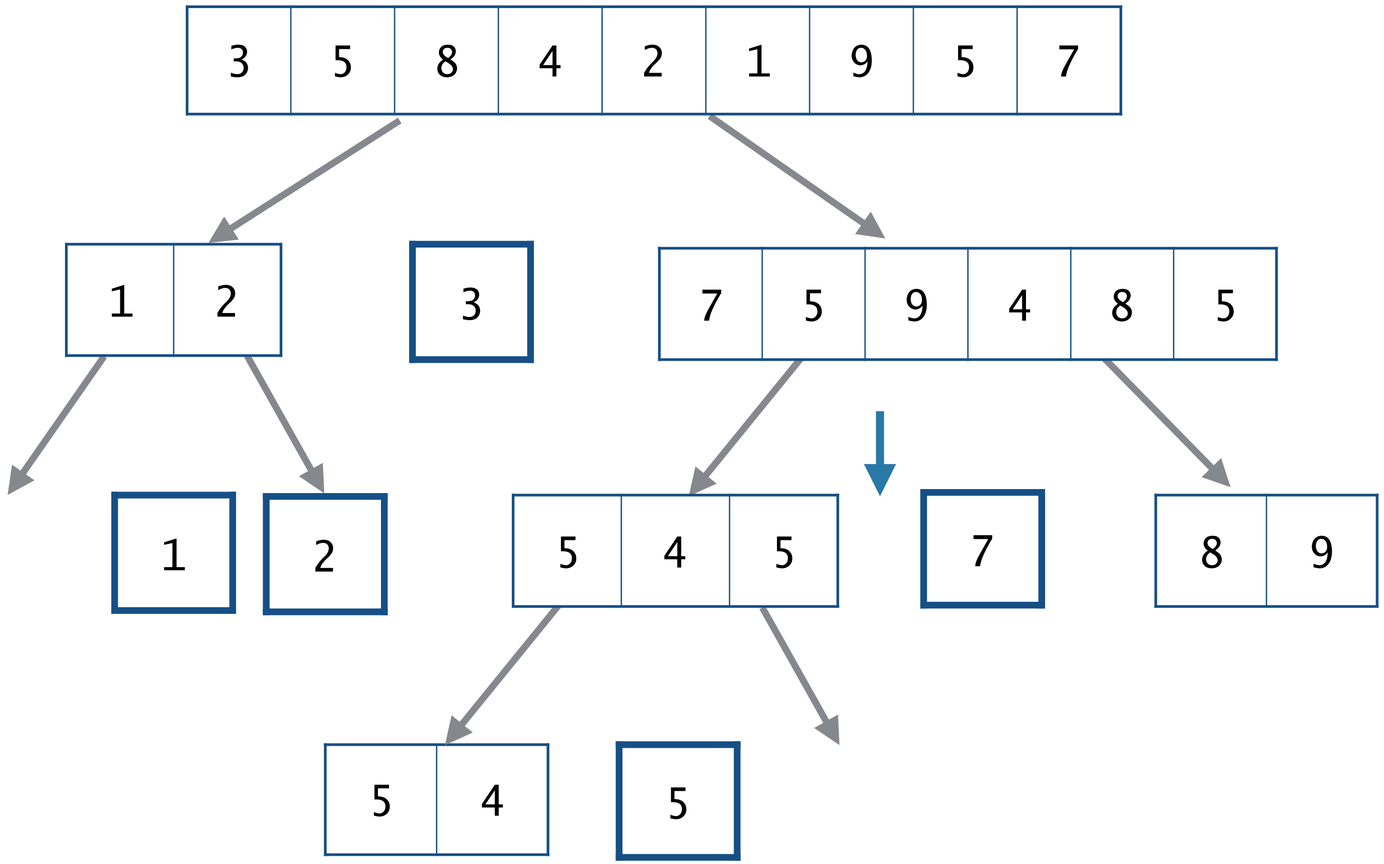
퀵정렬

Quicksort



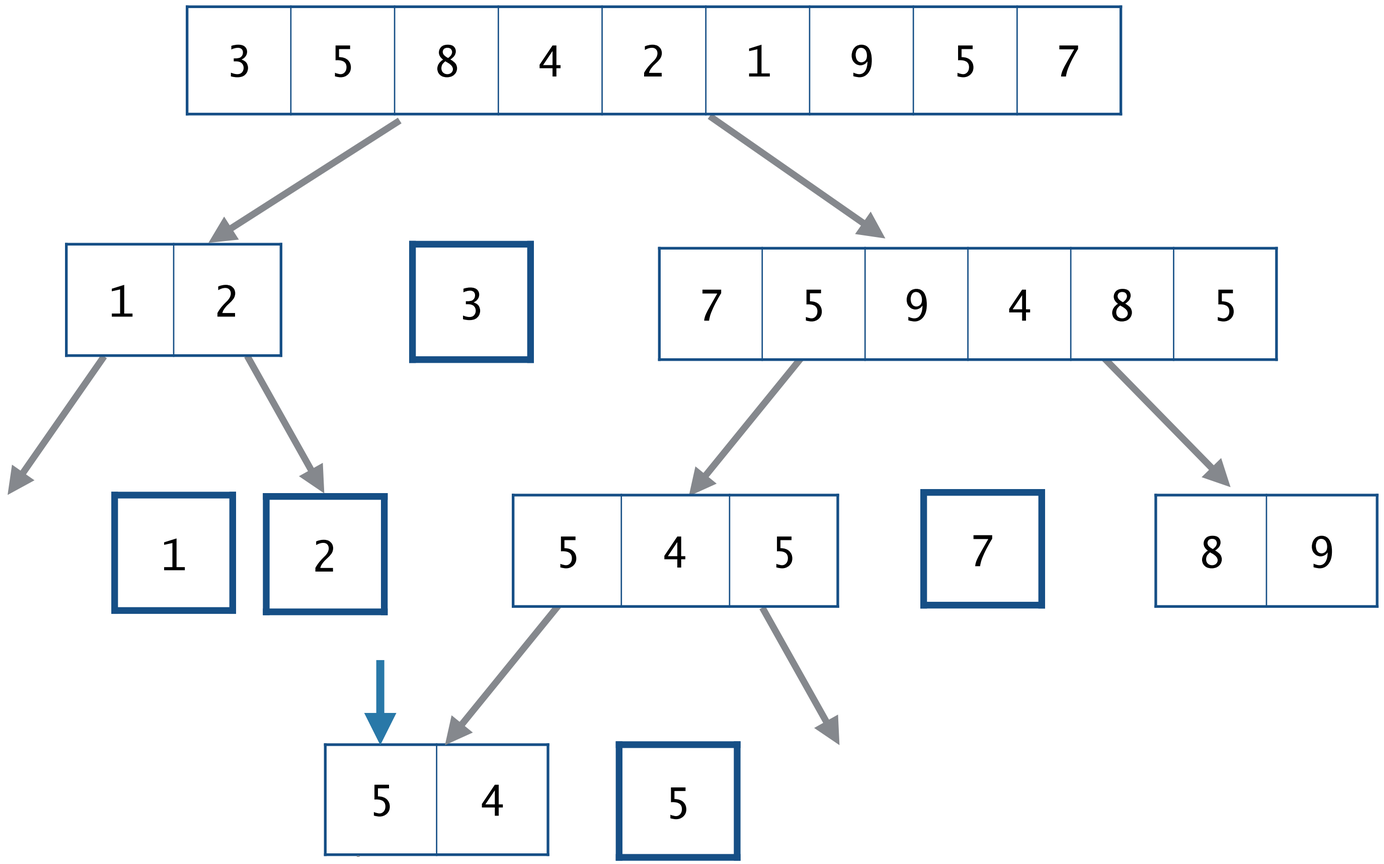
퀵정렬

Quicksort



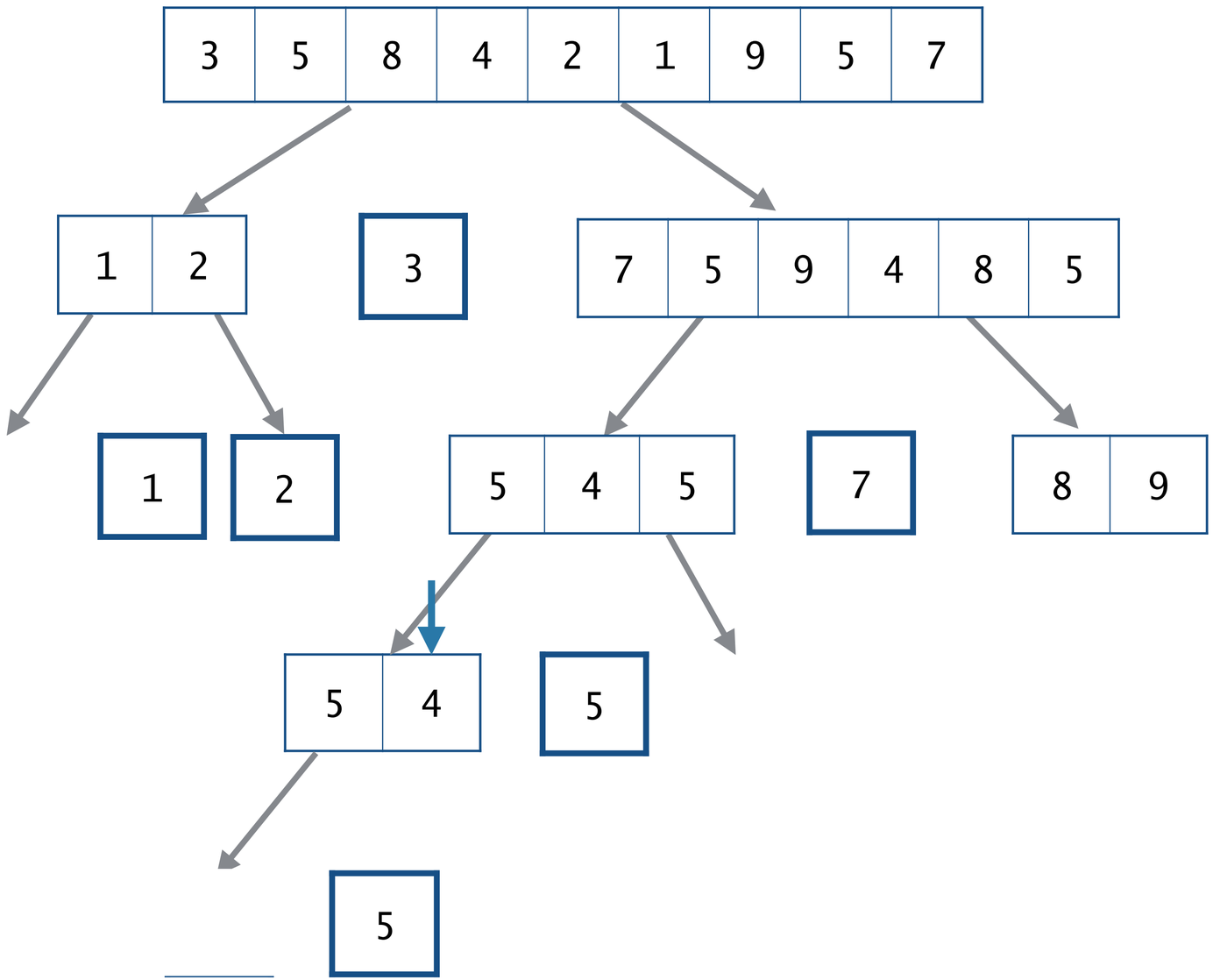
퀵정렬

Quicksort



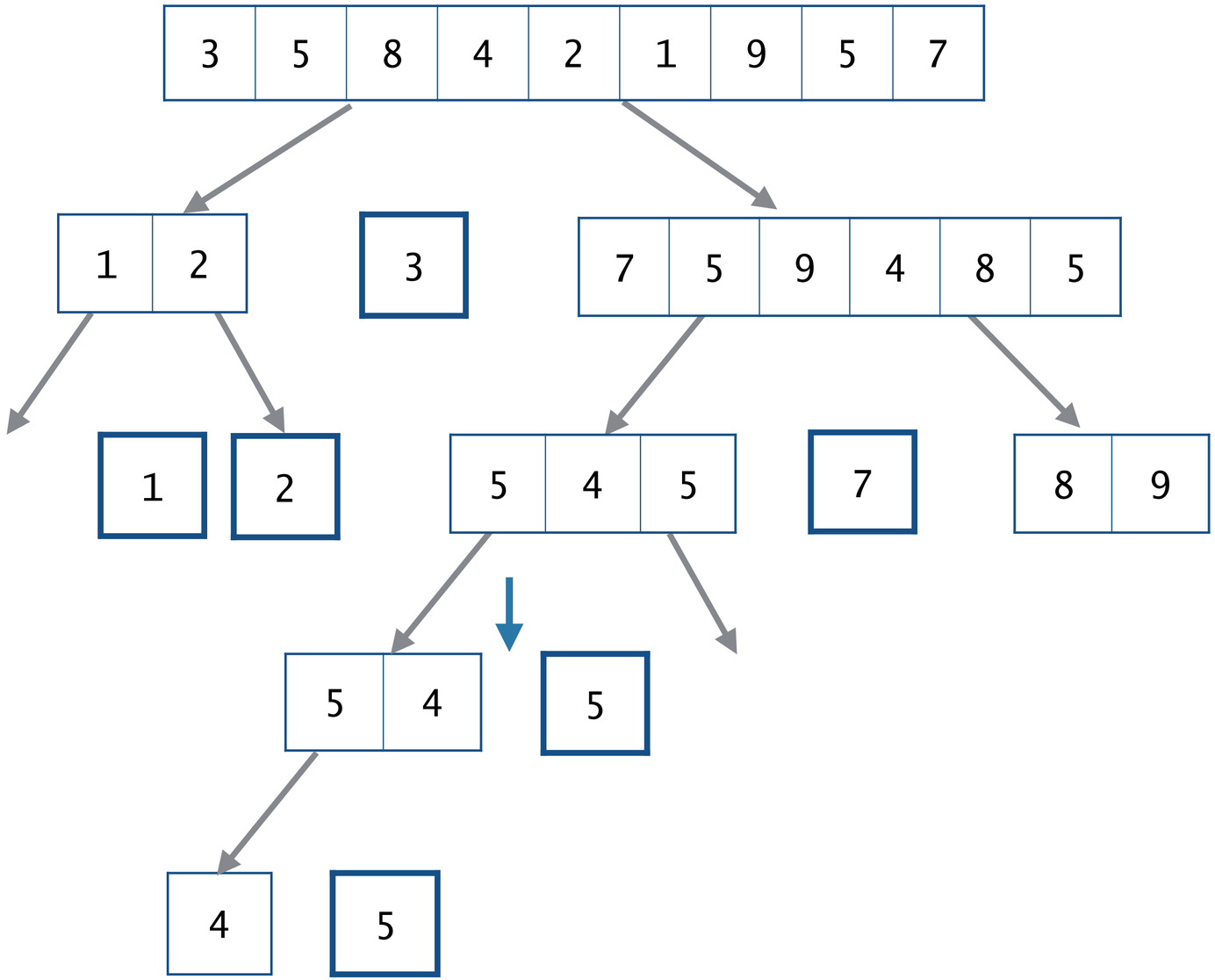
퀵정렬

Quicksort



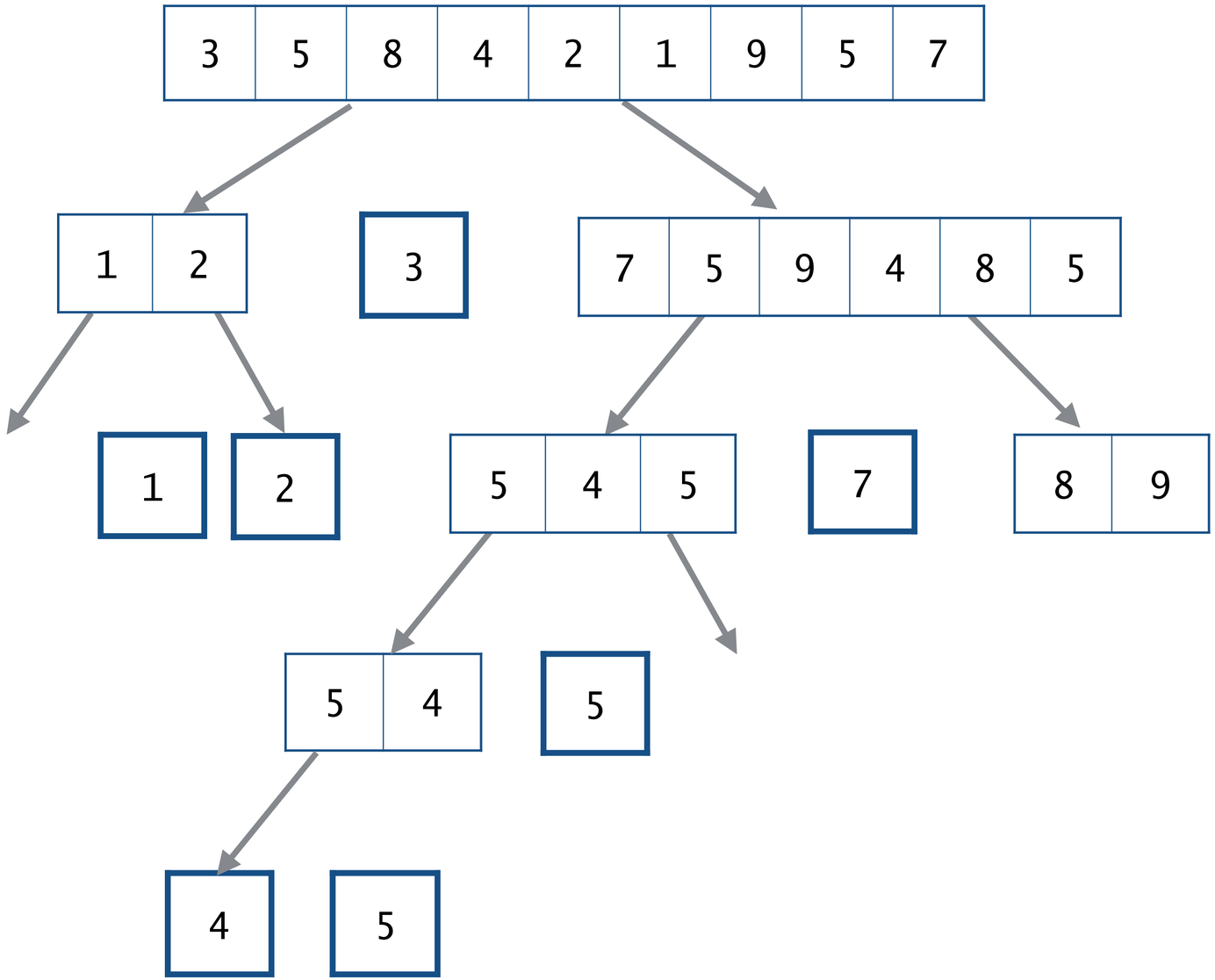
퀵정렬

Quicksort



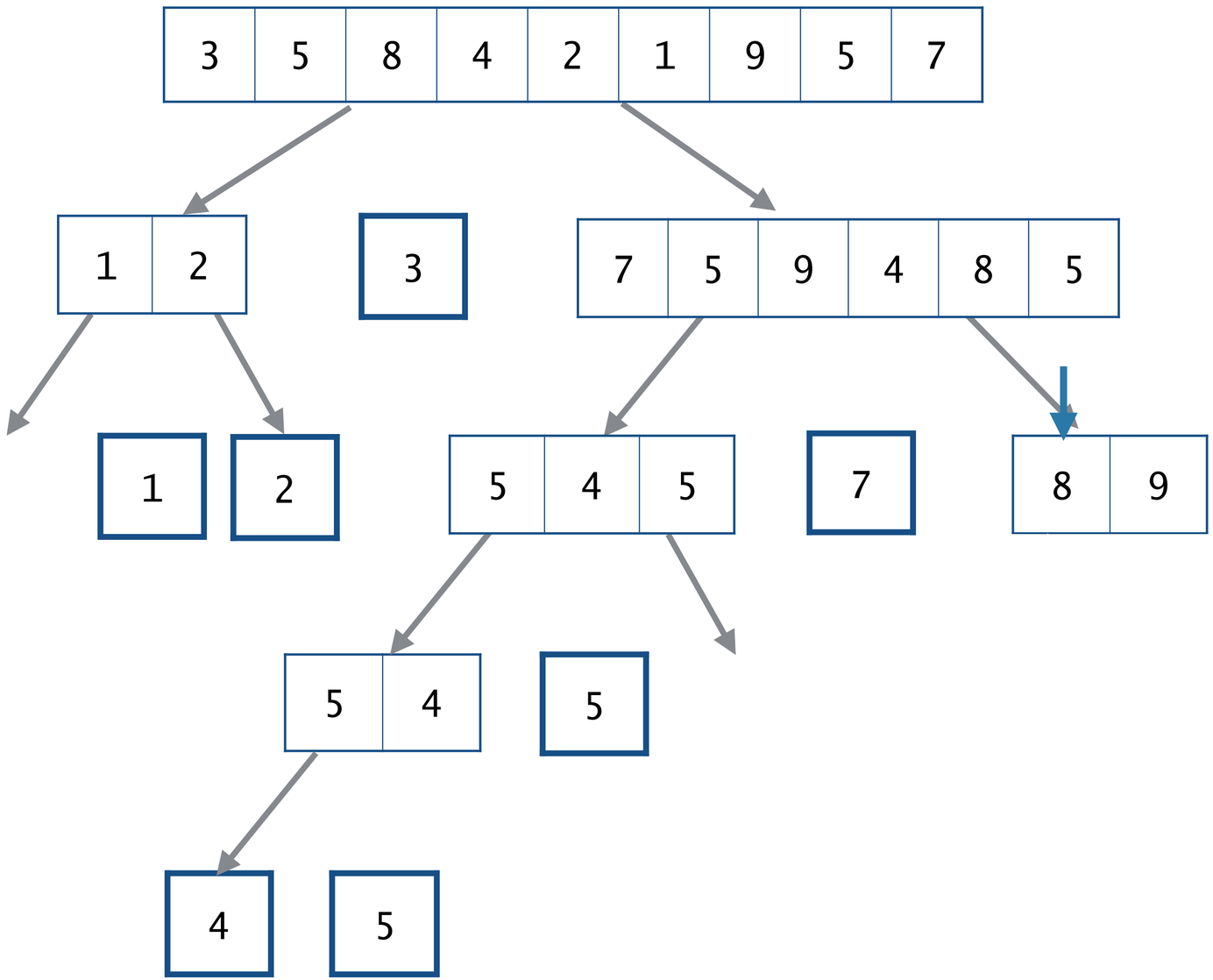
퀵정렬

Quicksort



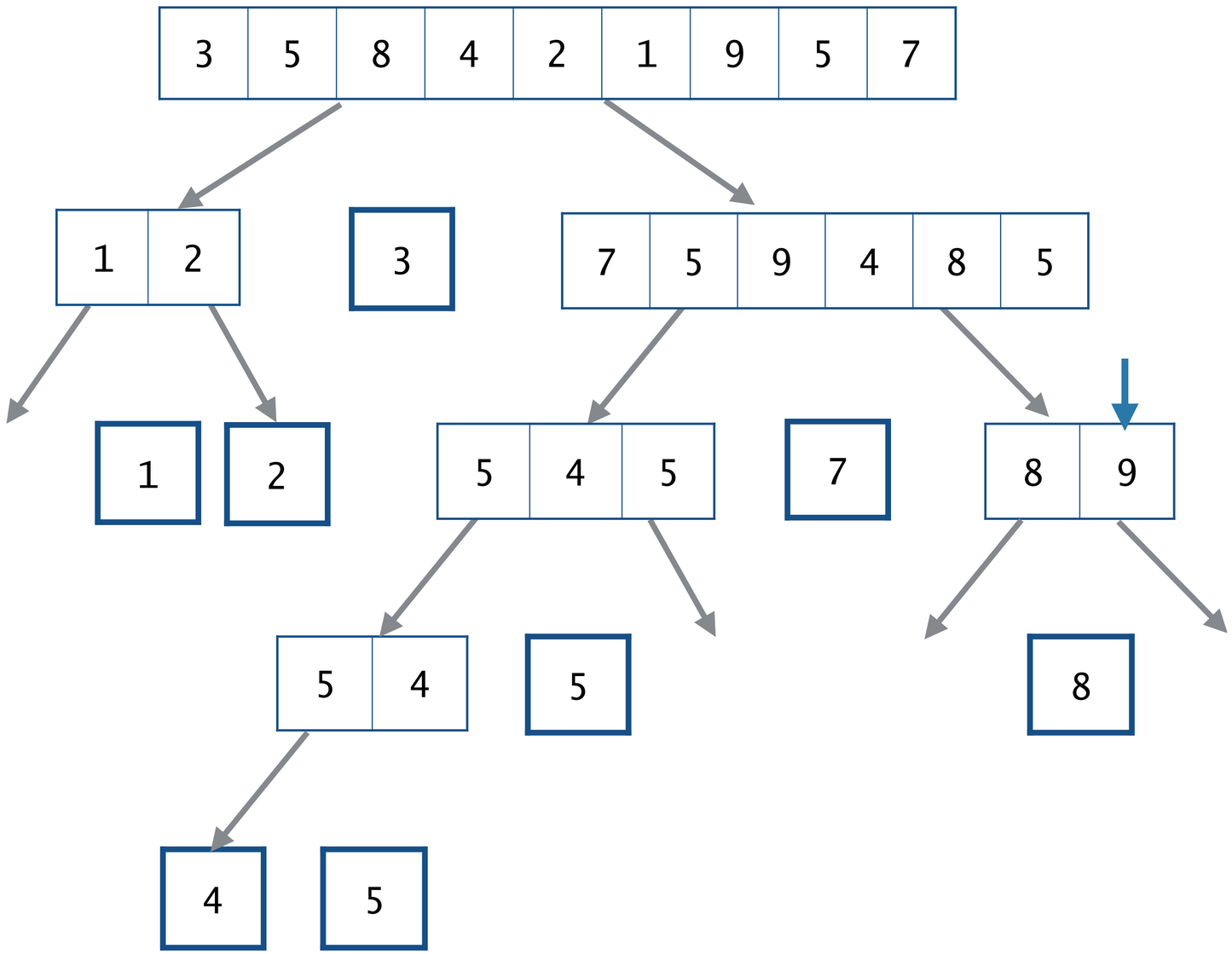
퀵정렬

Quicksort



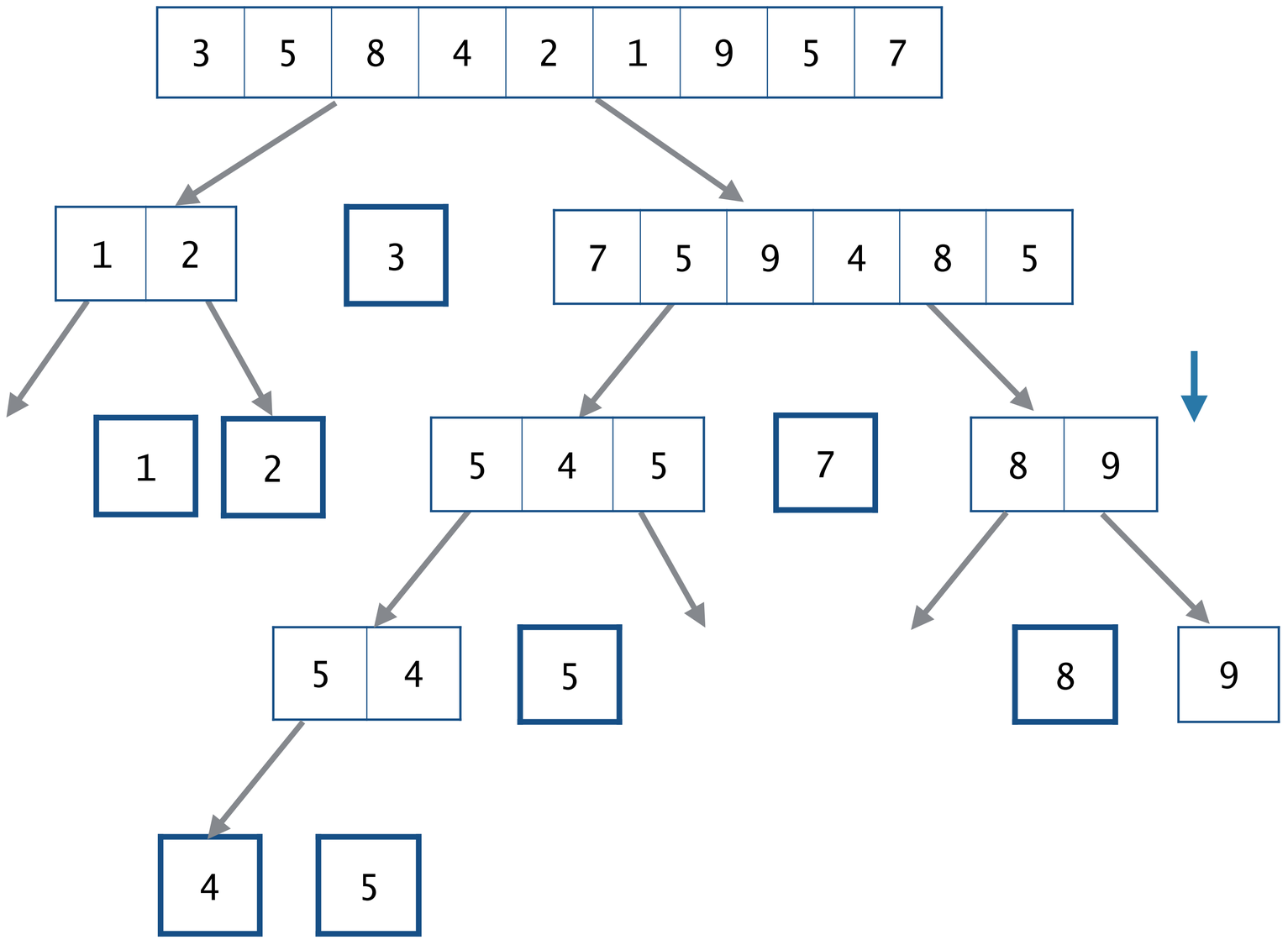
퀵정렬

Quicksort



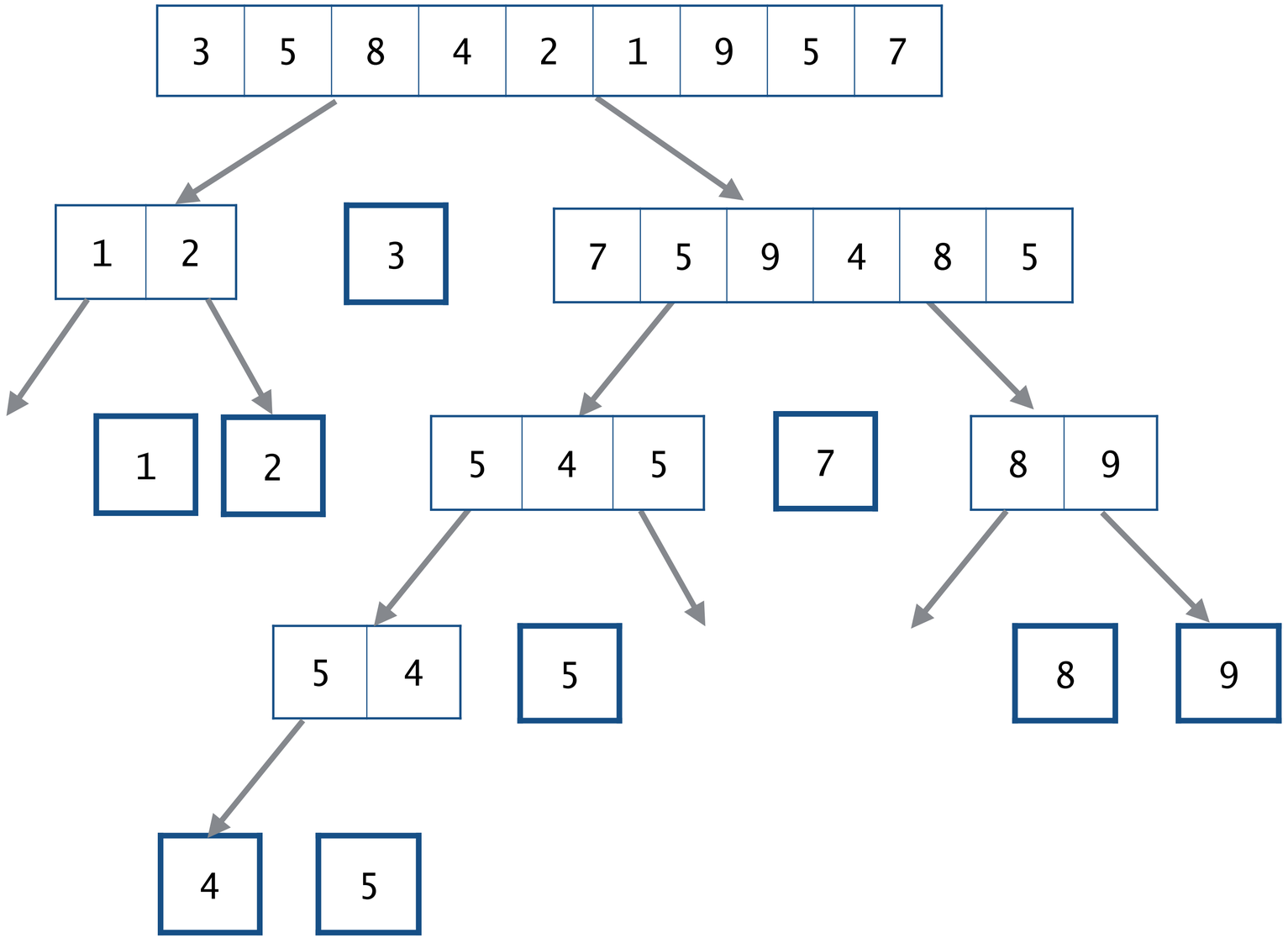
퀵정렬

Quicksort



퀵정렬

Quicksort



리스트 xs를 퀵정렬하려면		
반복 조건	len(xs) > 1	• 기준으로 사용할 피봇 원소 pivot을 하나 고른다. 편의상 맨 앞에 있는 원소를 고르기로 한다. • pivot을 기준으로 작은 원소는 왼쪽 리스트 left로, 큰 원소는 오른쪽 리스트 right로 옮긴다. • 왼쪽 리스트 left와 오른쪽 리스트 right를 각각 재귀로 정렬하고, left와 pivot과 right를 나란히 붙여서 리턴한다.
종료 조건	len(xs) <= 1	• 정렬할 필요가 없으므로 그대로 리턴한다.

code : 5-28.py

```
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
```

code : 5-29.py

```
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 [], []
```

```
partition(5,[7,2,1,9,4])
=>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    =>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       =>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          =>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             =>
```

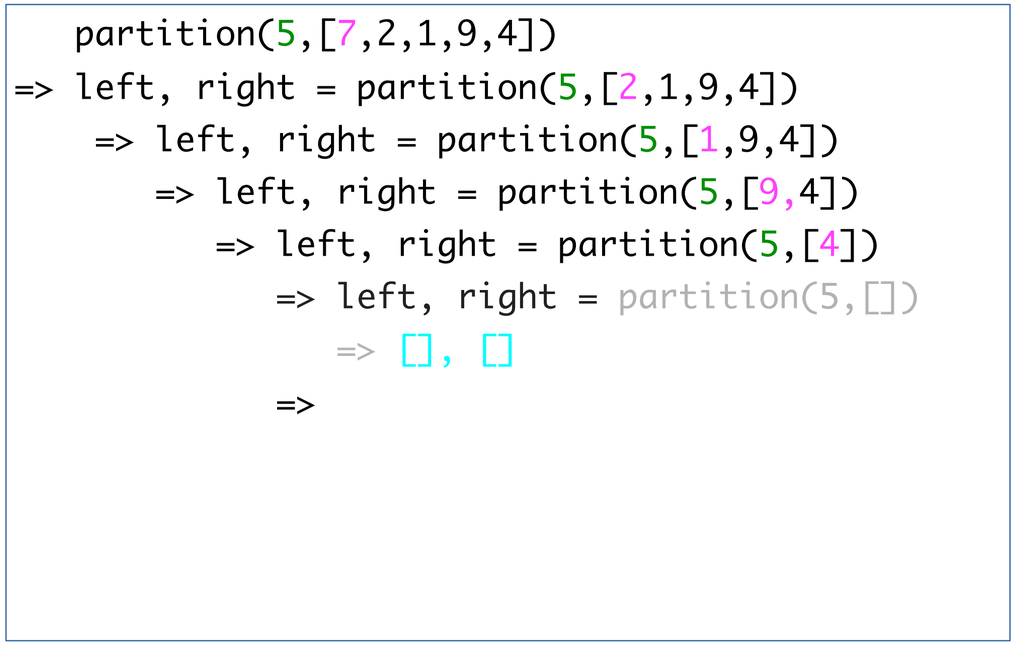
code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             => left, right = partition(5,[])
                =>
```

code : 5-29.py

```
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 [], []
```



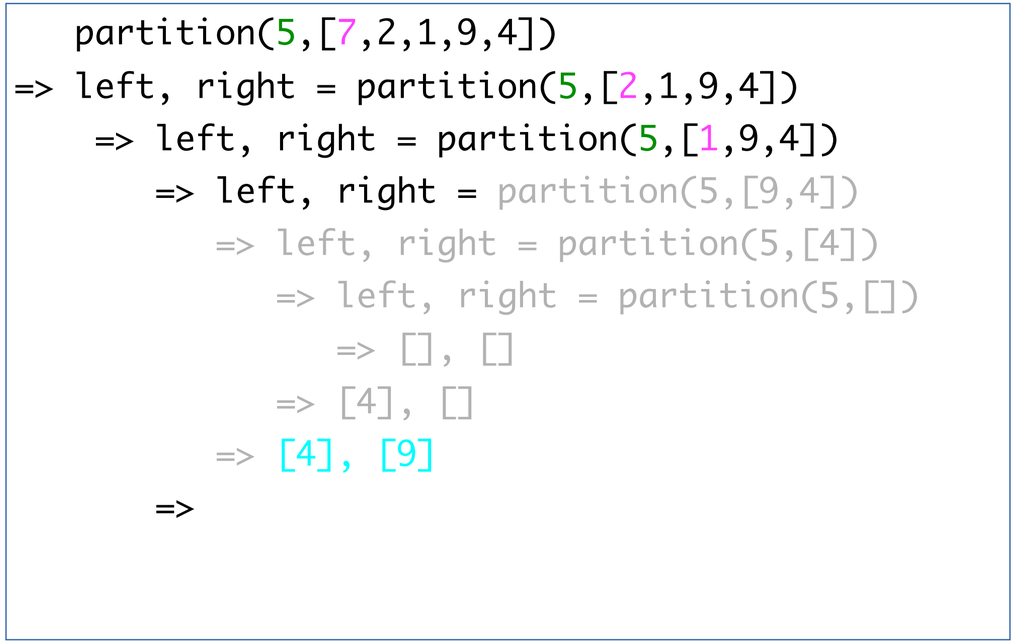
code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             => left, right = partition(5,[])
                => [], []
             => [4], []
          =>
```

code : 5-29.py

```
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 [], []
```



code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             => left, right = partition(5,[])
                => [], []
             => [4], []
          => [4], [9]
       => [4,1], [9]
    =>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             => left, right = partition(5,[])
                => [], []
             => [4], []
          => [4], [9]
       => [4,1], [9]
    => [4,1,2], [9]
=>
```

code : 5-29.py

```
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 [], []
```

```
   partition(5,[7,2,1,9,4])
=> left, right = partition(5,[2,1,9,4])
    => left, right = partition(5,[1,9,4])
       => left, right = partition(5,[9,4])
          => left, right = partition(5,[4])
             => left, right = partition(5,[])
                => [], []
             => [4], []
          => [4], [9]
       => [4,1], [9]
    => [4,1,2], [9]
=> [4,1,2], [9,7]
```

pp.247~248

실습 5.10 partition : 꼬리재귀 함수 버전

위의 partition 함수는 꼬리재귀가 아니다. 다음 뼈대코드에 맞추어 빈칸을 채워서 꼬리재귀 함수로 변환하자.

code : 5-30.py

```
1  def partition(pivot,xs):
2      def loop(xs,left,right):
3          if xs != []:
4              pass # Write your code here.
5
6
7
8
9          else:
10             return left, right
11     return loop(xs,[],[])
```

실습 5.11 partition : while 루프 버전

위의 partition 함수의 꼬리재귀 버전을 참조하면서, 다음 뼈대코드의 빈칸을 채워 while 루프 버전을 작성하자.

code : 5-31.py

```
1  def partition(pivot,xs):
2      left, right = [], []
3      while xs != []:
4          pass # Write your code here.
5
6
7
8
9      return left, right
```

실습 5.12 partition : for 루프 버전

partition 함수는 for 루프를 사용하면 훨씬 더 명료하게 작성할 수 있다. for 루프 버전을 작성하자.

code : 5-32.py

```
1  def partition(pivot,xs):
2      left, right = [], []
3      for x in xs:
4          pass # Write your code here.
5
6
7
8      return left, right
```

이제 partition 함수를 완성했으니, quicksort 함수를 테스트해볼 수 있다. 잘 작동하는지 확인해 보자.

>>>>>>>>>>> 제어 구조의 설계 원리를 중심으로 배우는 >>>>>>>>>>>

프로그래밍의 정석

파이썬

도경구 지음

생능출판

CHAPTER 5

재귀와 반복 : 정렬