

CES1017 Programming Fundamentals

Week 6: Recursion and Iteration - Sorting

Instructor: Eunil Park (pa1324@hanyang.ac.kr)



HANYANG UNIVERSITY



Last Class

1. 자연수
2. 첫째 사례: 초 읽기 출력
3. 둘째 사례: 1부터 n 까지 자연수 합 계산
4. 셋째 사례: b^n 계산

Sorting

- 순서를 매길 수 있는 데이터를, 정한 순서로, 나열하는 문제
 - 예: [3, 5, 2, 4] → [2, 3, 4, 5]
- Sorting은 CS에서 전통적으로 중요한 문제
 - 정렬된 데이터의 처리가 훨씬 효율적임
 - 수많은 sorting 알고리즘이 있음

Today's Schedule

1. 순서열
2. 선택정렬
3. 삽입정렬
4. 합병정렬
5. 퀵정렬
6. 버블정렬

순서열(sequence)

- 여러 개의 데이터를 순서있게 나열해 모아놓은 데이터 구조
- Python에서 기본 제공하는 순서열
 - 리스트(list)
 - 튜플(tuple)
 - 정수 범위(range)
 - 문자열
- 순서열은 index (위치번호)로 접근 가능
- 수정가능(mutable) 순서열
 - List
- 수정불가능(immutable) 순서열
 - Tuple, range, 문자열

리스트(list)

- 대괄호[]로 표현

```
>>> odds = [1, 3, 5, 7, 9]
```

```
>>> odds
```

```
[1, 3, 5, 7, 9]
```

```
>>> odds[1]
```

```
3
```

```
>>> odds[-2]
```

```
7
```

```
>>> odds[5]
```

odds



1	3	5	7	9
---	---	---	---	---

01. 순서열



리스트 수정/복제

- `odds[3] = 77`
- `odds[1:3] = 33, 55`
- `odds[1:3] = 34, 44, 55`

`odds` →

1	3	5	77	9
---	---	---	----	---

`odds` →

1	33	55	77	9
---	----	----	----	---

`odds` →

--	--	--	--	--

- `odds2 = odds`
- `odds2[2] = 4`

`odds` →

1	34	44	55	77	9
---	----	----	----	----	---

`odds2` →

1	34	4	55	77	9
---	----	---	----	----	---

`odds` →

1	34	4	55	77	9
---	----	---	----	----	---

`odds2` →

1	34	4	55	77	9
---	----	---	----	----	---

- `#list copy`
`odds2 = odds[:]`

`odds` →

1	34	4	55	77	9
---	----	---	----	----	---

`odds2` →

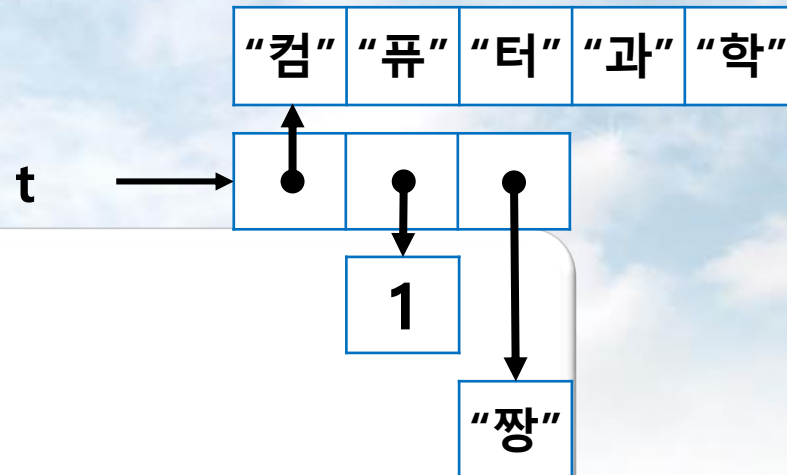
1	34	4	55	77	9
---	----	---	----	----	---

튜플(tuple)

- 괄호로 표현

```
>>> t = ('컴퓨터과학', 1, '짱')  
>>> t[2]  
'짱'  
>>> t[:2]  
('컴퓨터과학', 1)  
>>> t[2] = '짱'
```

```
>>> t[1] = 3
```

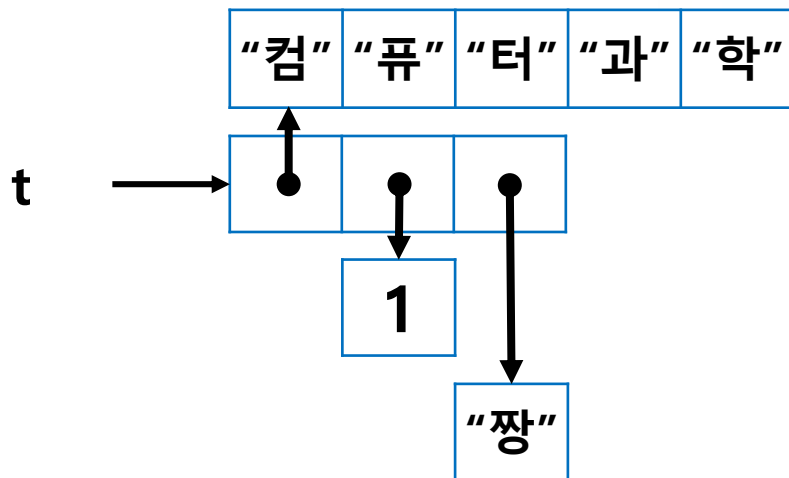


새 튜플 생성

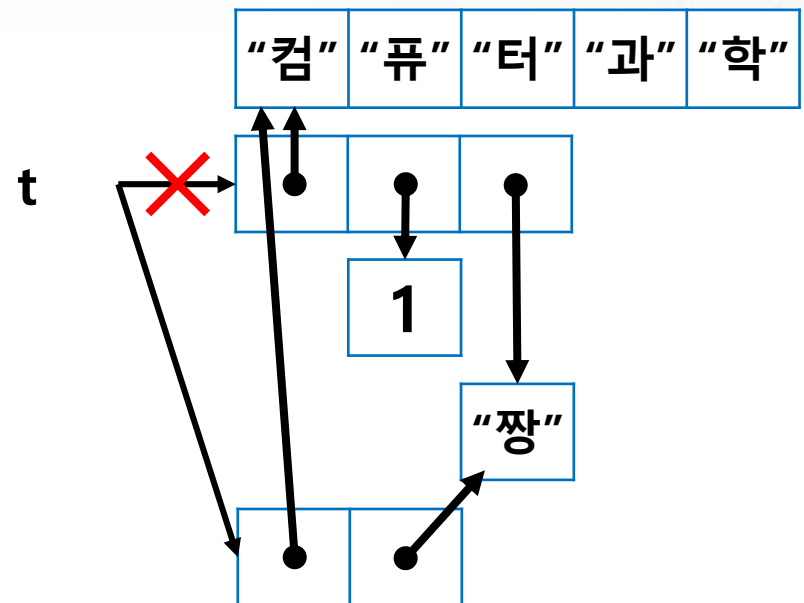
- 새 튜플 "t" 생성(수정x)

$t = t[:1] + t[2:]$

[실행 전]



[실행 후]



수정이 잦은 데이터는 리스트에 보관하는 것이 효율적일 수 있음

문자열(string)

- 수정 불가능(immutable)

```
>>> s = "컴퓨터과학"
>>> s
'컴퓨터과학'
>>> s[3] = '공'
```

s → "컴" "퓨" "터" "과" "학"

- 수정이 불가능하기 때문에 새로운 문자열 생성만 가능

```
>>> s = s + '도'
```

[실행 전] s → "컴" "퓨" "터" "과" "학"

정수범위(range)

- 정수가 일정 간격으로 나열된 순서열
- `range(n)`은 0부터 $n-1$ 까지 간격 1의 정수열
- 예: `range(5)`는 0, 1, 2, 3, 4 순서열

```
>>> r = range(5)
>>> r
range(0, 5)
>>> r[2]
2
>>> r[2] = 3
```

```
>>> 3 in r
True
>>> 5 in r
False
```

range(m,n,k)

- 정수 m부터 n-1까지 간격 k의 정수 범위 순서열
- 예:

range(3,10) = 3, 4, 5, 6, 7, 8, 9, 10

range(3,11,2) =

range(10,3,-1) =

반복문: for

- 길이가 고정된 순서열을 하나씩 처리하는 경우 편리함
- 문법

문법	for <변수 이름> in <순서열>: <몸체>
의미	<변수 이름>을 x라고 하고, <순서열>을 s라고 하면, 다음을 차례대로 실행한다. <ul style="list-style-type: none">- s[0]를 변수 x로 지정하고 <몸체> 를 실행한다.- s[1]를 변수 x로 지정하고 <몸체> 를 실행한다.- s[2]를 변수 x로 지정하고 <몸체> 를 실행한다.-- s[len(s)-1]를 변수 x로 지정하고 <몸체> 를 실행한다.

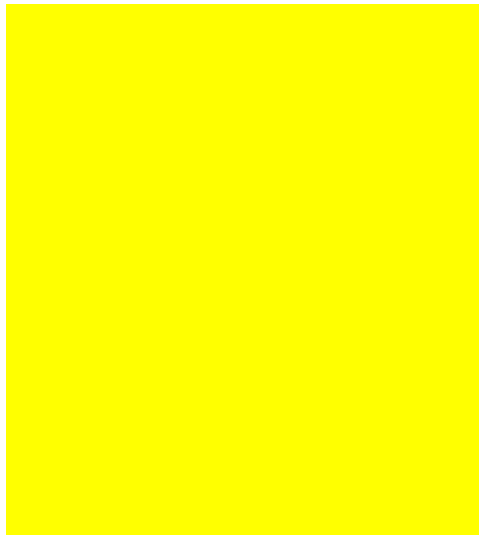
01. 순서열



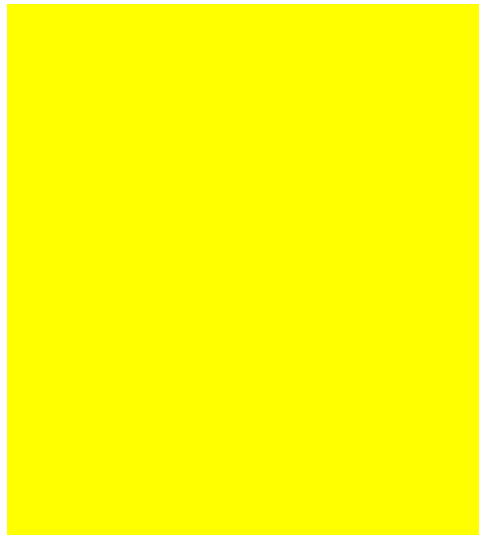
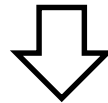
반복문: for

- 예제

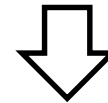
```
for i in range(5):  
    print(i)
```



```
for i in range(3,10):  
    print(i)
```



```
for i in range(10,3,-3):  
    for j in range(3,11,3):  
        print(i, j)
```



01. 순서열



순서열 연산

• $s = [1, 4, 7, 1, 5, 1, 4, 7]$

$s \rightarrow$

1	4	7	1	5	1	4	7
---	---	---	---	---	---	---	---

연산	의미	예시	결과
$x \text{ in } s$	x 가 s 에 있으면 True, 없으면 False	$3 \text{ in } s$	
$x \text{ not in } s$	x 가 s 에 없으면 True, 있으면 False	$9 \text{ not in } s$	
$s[i]$	s 의 i 위치에 있는 원소	$s[6]$	
$s[i:j]$	i 위치(포함)에서 j 위치(제외)까지 순서열 조각	$s[2:7]$	
$s[i:j:k]$	i 위치(포함)에서 j 위치(제외)까지의 k 간격으로 띄운 순서열 조각	$s[2:8:2]$	
$\text{len}(s)$	s 의 길이	$\text{len}(s)$	
$\text{min}(s)$	s 에서 가장 작은 원소	$\text{min}(s)$	

01. 순서열



순서열 연산

• $s = [1, 4, 7, 1, 5, 1, 4, 7]$

$s \rightarrow$

1	4	7	1	5	1	4	7
---	---	---	---	---	---	---	---

연산	의미	예시	결과
$\max(s)$	s에서 가장 큰 원소	$\max(s)$	
$s.\text{index}(x)$	s에서 가장 앞에 나오는 원소 x의 위치번호	$s.\text{index}(7)$	
$s.\text{index}(x, i)$	s의 i위치에서 시작하여 가장 앞에 나오는 원소 x의 위치번호	$s.\text{index}(1, 3)$	
$s.\text{index}(x, i, j)$	s의 i 위치로부터 j 위치 전까지에서 가장 앞에 나오는 원소 x의 위치번호	$s.\text{index}(1, 4, 7)$	
$s.\text{count}(x)$	s에서 원소 x의 빈도수	$s.\text{count}(1)$	
$s + t$	s와 t 나란히 붙이기	-	
$s * n$	s를 n번 반복하여 나란히 붙이기	-	
$n * s$	s를 n번 반복하여 나란히 붙이기	-	

리스트의 귀납 정의

(1) 기초 (basis) 빈 리스트 $[]$ 은 리스트이다.

(2)	귀납 (induction)	x 가 임의의 원소이고, L 이 임의의 리스트이면, $[x] + L$ 도 리스트이다.
-----	----------------	---

(3) 그 외에 다른 리스트는 없다.

02. 선택정렬(selection sort)



알고리즘

정수 리스트 s 를 정렬하려면...?

- (반복조건) $s \neq []$,

1.

2.

3.

- (종료조건) $s == []$, 정렬할 필요가 없으므로 $[]$ 를 그대로 내줌

02. 선택정렬(selection sort)



구현

```
1 def ssort0(s):
2     if s != []:
3         smallest = min(s)
4         s.remove(smallest)
5     return [smallest] + ssort0(s)
6 else:
7     return []
```

연산	의미	비고
s.remove(x)	s에서 가장 앞에 나오는 원소 x를 제거한다.	x가 s에 없으면 ValueError가 발생한다.
1. s에서 가장 작은 수를 찾아서 smallest 라고 하고, 이를 s에서 제거	2. s를 정렬	3. smallest와 s를 차례로 나란히 붙여서 내쫓 + 재귀를 활용한 반복
s == [], 정렬할 필요가 없으므로 []를 그대로 내쫓		

- ssort0([3, 5, 4, 2])

02. 선택정렬(selection sort)



주의할 점

```
1 def ssort0err(s):
2     if s != []:
3         smallest = min(s)
4         return [smallest] + ssort0err(s.remove(smallest))
5     else:
6         return []
```

오류!

연산	의미	비고
s.remove(x)	s에서 가장 앞에 나오는 원소 x를 제거한다.	x가 s에 없으면 ValueError가 발생한다.

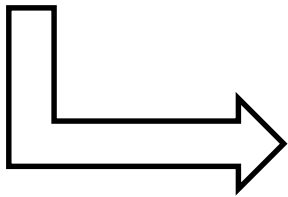
s.remove(x)가 리스트를 리턴해 주는 것이 아님

02. 선택정렬(selection sort)



꼬리재귀

```
1 def ssort0(s):  
2     if s != []:  
3         smallest = min(s)  
4         s.remove(smallest)  
5         return [smallest] + ssort0(s)  
6     else:  
7         return []
```



```
1 def ssort1(s):  
2  
3  
4  
5  
6  
7  
8  
9
```

02. 선택정렬(selection sort)



append() 사용

```
1 def ssort1(s):
2     def loop(s, ss):
3         if s != []:
4             smallest = min(s)
5             s.remove(smallest)
6             return loop(s, ss+[smallest])
7         else:
8             return ss
9     return loop(s, [])
```

연산	의미	비고
s.append(x)	s의 맨 뒤에 x를 붙인다.	

```
1 def ssort2(s):
2     def loop(s, ss):
3         if s != []:
4             smallest = min(s)
5             s.remove(smallest)
6             # (lines 6-7 are highlighted yellow in the original image)
7
8         else:
9             return ss
10    return loop(s, [])
```

02. 선택정렬(selection sort)



While반복문

```
1 def ssort3(s):  
2   
3   
4   
5   
6   
7 return ss
```

```
def ssort2(s):  
    def loop(s, ss):  
        if s != []:  
            smallest = min(s)  
            s.remove(smallest)  
            ss.append(smallest)  
            return loop(s, ss)  
        else:  
            return ss  
    return loop(s, [])
```

03. 삽입정렬(Insertion sort)



알고리즘

정수 리스트 s 를 정렬하려면...?

- (반복조건) $s \neq []$,
 1. s 의 후미리스트인 $s[1:]$ 를 정렬하여 ss 라고 함
 2. 정렬된 리스트 ss 의 적당한 위치에 s 의 선두원소인 $s[0]$ 를 끼워서 내줌
- (종료조건) $s == []$, 정렬할 필요가 없으므로 그대로 내줌

03. 삽입정렬(Insertion sort)

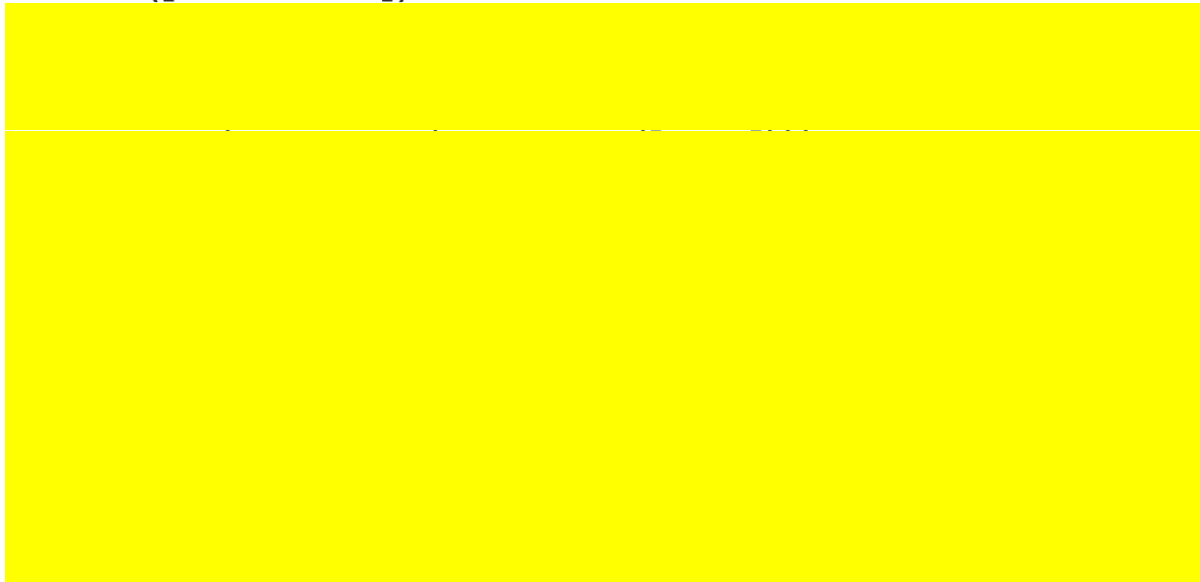


구현

```
1 def isort0(s):  
2     if s != []:  
3         return insert(s[0], isort(s[1: ]))  
4     else:  
5         return []
```

- 정렬된 리스트 ss의 적절한 위치에 x를 끼워서 정렬된 리스트를 내어주는 insert(x,ss)가 있다고 가정

- isort0([3, 5, 4, 2])



03. 삽입정렬(Insertion sort)



insert 함수 구현

예제:

- $\text{insert}(1, [2, 4, 5, 7, 8]) \rightarrow [1, 2, 4, 5, 7, 8]$
- $\text{insert}(6, [2, 4, 5, 7, 8]) \rightarrow [2, 4, 5, 6, 7, 8]$
- $\text{insert}(9, [2, 4, 5, 7, 8]) \rightarrow [2, 4, 5, 7, 8, 9]$

03. 삽입정렬(Insertion sort)



재귀 함수 설계 전략

사례를 통한 구상:

- 정렬된 리스트 ss의 선두원소보다 작은 수 끼워넣기

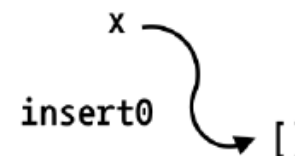
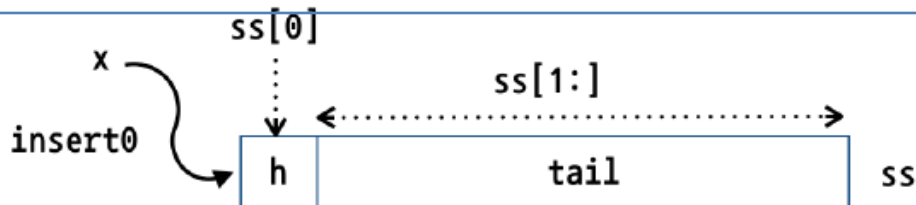
$\text{insert}(1, [2, 4, 5]) \rightarrow [1] + [2, 4, 5]$

- ss의 선두원소보다 큰 수 끼워넣기

$\text{insert0}(3, [2, 4, 5]) \rightarrow [2] + \text{insert0}(3, [4, 5])$

- 빈 리스트에 끼워넣기

$\text{insert0}(1, []) \rightarrow [1]$



- $x \leq h$



- $x > h$



03. 삽입정렬(Insertion sort)



insert0(x,ss) 알고리즘

- ss가 빈 리스트가 아닌 경우,
 1. ss를 선두원소 ss[0]와 후미리스트 ss[1:]로 나눈다.
 2. x가 선두원소 ss[0]보다 작거나 같으면, x를 ss의 앞에 붙인다.
 3. x가 선두원소 ss[0]보다 크면, x를 후미리스트 ss[1:]의 제 위치에 끼워넣고 그 앞에 선두원소 ss[0]를 붙인다.
- ss가 빈 리스트인 경우, 그냥 x만 가지고 리스트를 만든다.

[실행 예시]

- insert0(1, [2, 4, 5, 7, 8])
→ [1, 2, 4, 5, 7, 8]
- insert0(6, [2, 4, 5, 7, 8])
- insert0(9, [])
→ [9]

04. 합병정렬(Merge sort)

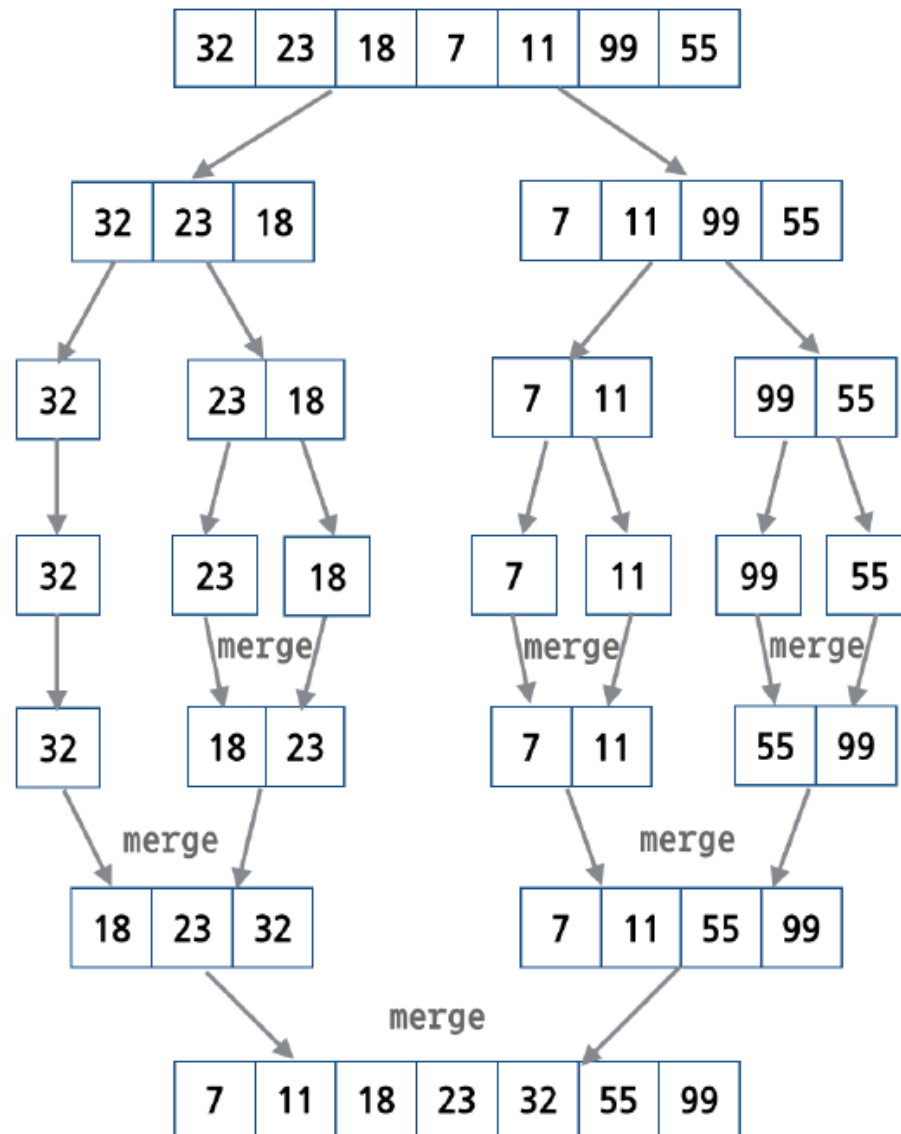


알고리즘

- $\text{len}(s) > 1$
 - s 를 반으로 쪼개서, 따로 재귀로 정렬을 완료하고,
두 정렬된 리스트를 앞에서부터 차례로 훑어가며, 가장 작은 수를 먼저
선택하는 방식으로 하나로 합병 (merge)한다.
- $\text{len}(s) \leq 1$
 - 정렬할 필요가 없으므로 그대로 내준다.
- 실행 예 – [32, 55, 18, 7, 11, 99, 23]
 1. [32, 55, 18]과 [7, 11, 99, 23]으로
반으로 쪼개서 각각 합병정렬한다.
 2. 정렬된 두 리스트 [18, 32, 55]와 [7, 11, 23, 99]를 앞에서 훑어가며
가장 작은 수를 먼저 선택하는 방식으로 합병한다.

04. 합병정렬(Merge sort)

도식화



04. 합병정렬(Merge sort)



구현

```
1 def msort(s):  
2     if len(s) > 1:  
3           
4         return merge(  
5     else:  
6         return s
```

- $\text{len}(s) > 1$
 - s 를 반으로 쪼개서, 따로 재귀로 정렬을 완료하고, 두 정렬된 리스트를 앞에서부터 차례로 훑어가며, 가장 작은 수를 먼저 선택하는 방식으로 하나로 합병(merge)한다.
- $\text{len}(s) \leq 1$
 - 정렬할 필요가 없으므로 그대로 내준다.

04. 합병정렬(Merge sort)



merge 함수 구현

- 두 리스트 모두 정렬되어 있으므로, 각 리스트의 맨 앞에 있는 두 수 중에서 작은 수를 하나 빼내어 정렬된 리스트를 만들어나간다.
- 이 과정을 두 리스트 중에서 하나가 소진될 때까지 계속 반복한다.
- 둘 중에 하나가 소진되면 나머지를 정렬된 리스트 뒤에 붙인다.

```
1 def merge0(left, right):  
2     if  
3  
4  
5  
6  
7     e  
8
```

left != [] and right != []
→ 즉, left와 right 양쪽에
최소한 원소가 하나 있음

04. 합병정렬(Merge sort)



merge 사례



05. 퀵정렬(Quicksort)



알고리즘

- $\text{len}(s) > 1$
 1. 기준으로 사용할 피벗값 **pivot**을 하나 고른다. 편의상 맨 앞에 있는 수를 고르기로 한다.
(사실 피벗값을 잘 골라서 리스트가 항상 반으로 쪼개지는 경우, 퀵정렬의 성능이 가장 좋으므로 피벗값을 고르는 작업은 사실 중요하다.)
 2. **pivot**을 기준으로 작은 수는 왼쪽 리스트에, 큰 수를 오른쪽 리스트로 옮긴다.
 3. 왼쪽 리스트와 오른쪽 리스트를 각각 재귀로 정렬하고, 가운데에 **pivot**값을 끼운다.
- $\text{len}(s) \leq 1$
 - 정렬할 필요가 없으므로 그대로 내준다.

3	7	8	5	2	1	9	5	4
---	---	---	---	---	---	---	---	---

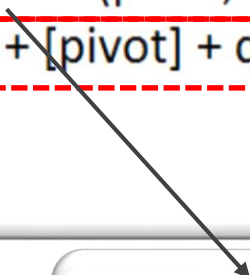
실행 예
[3, 7, 8, 5, 2, 1, 9, 5, 4]

05. 퀵정렬(Quicksort)



구현

```
def qsort(s):  
    if len(s) > 1:  
        pivot = s[0]  
        (left, right) = partition(pivot, s[1: ])  
        return qsort(left) + [pivot] + qsort(right)  
    else:  
        return s
```



```
1 def partition(pivot, s):  
2  
3     for x in s:  
4         if x <= pivot:  
5  
6     else:  
7  
8     return left, right
```

06. 버블정렬(Bubble sort)



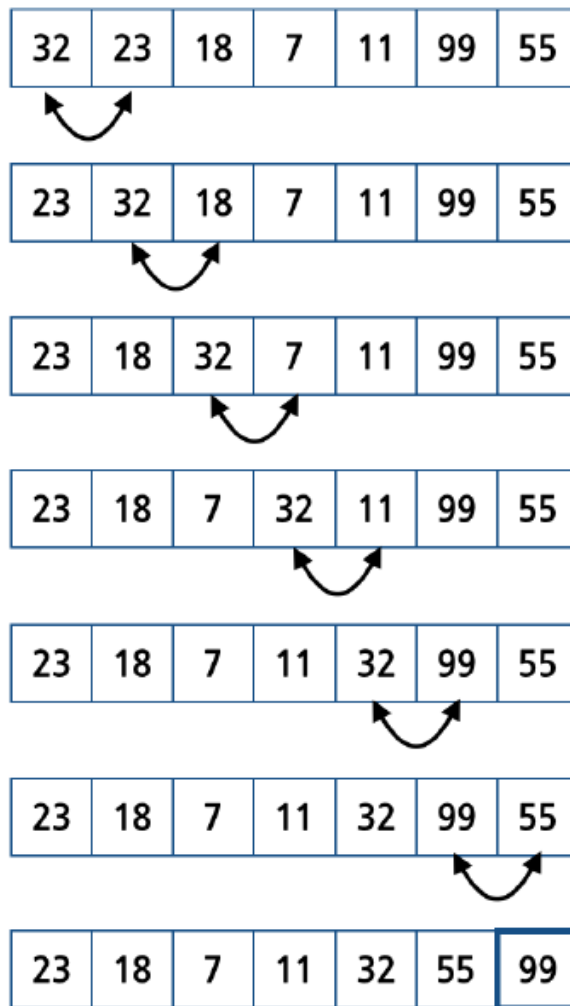
알고리즘

- 리스트에서 인접한 두 수를 비교하여 순서가 바뀐 경우 이를 교환 정렬함
- 제자리정렬(in-place sort)
 - 추가공간을 사용하지 않고 자체적으로 값을 교환 정렬
 - 버블정렬은 리스트 내부에서 수의 교환이 필요해서 추가 공간이 필요하지 않기 때문에 제자리정렬임
 - 선택정렬, 삽입정렬, 합병정렬, 퀵정렬은 제자리정렬이 아님

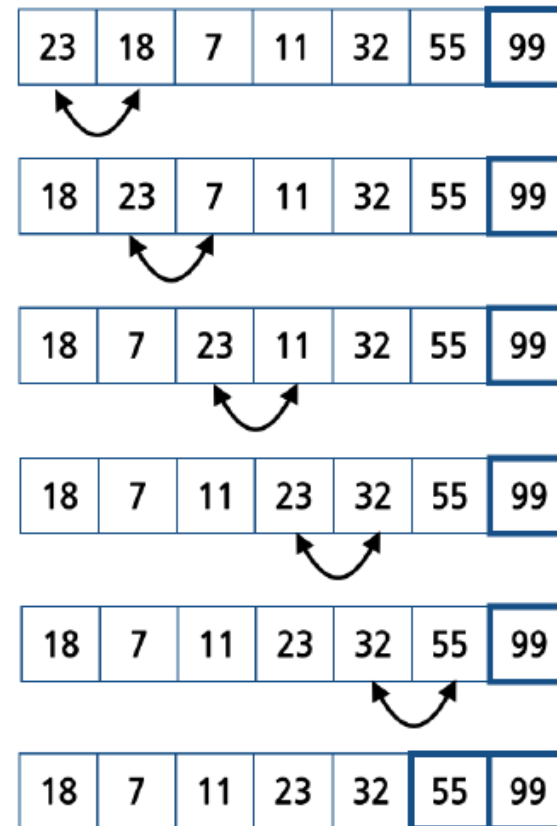
06. 버블정렬(Bubble sort)

실행 예

1st loop



2nd loop

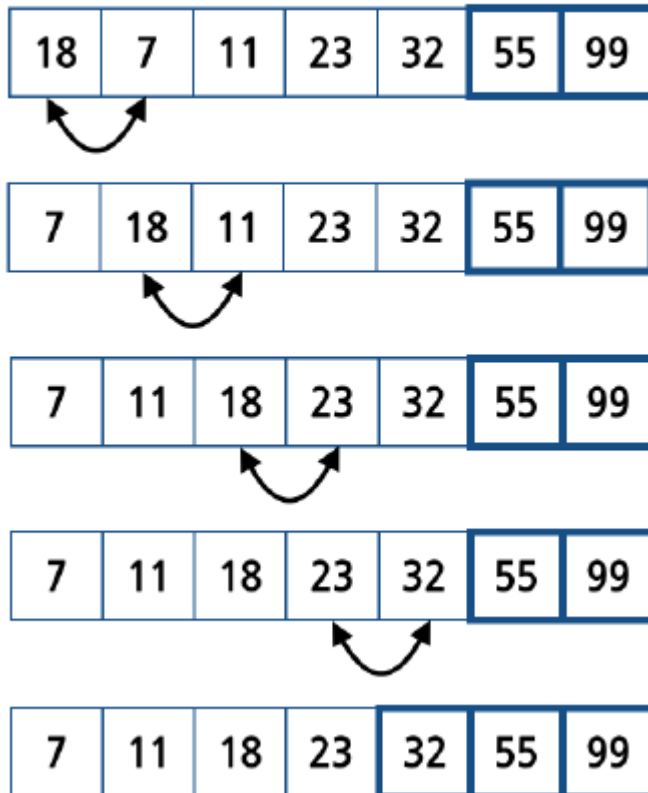


06. 버블정렬(Bubble sort)

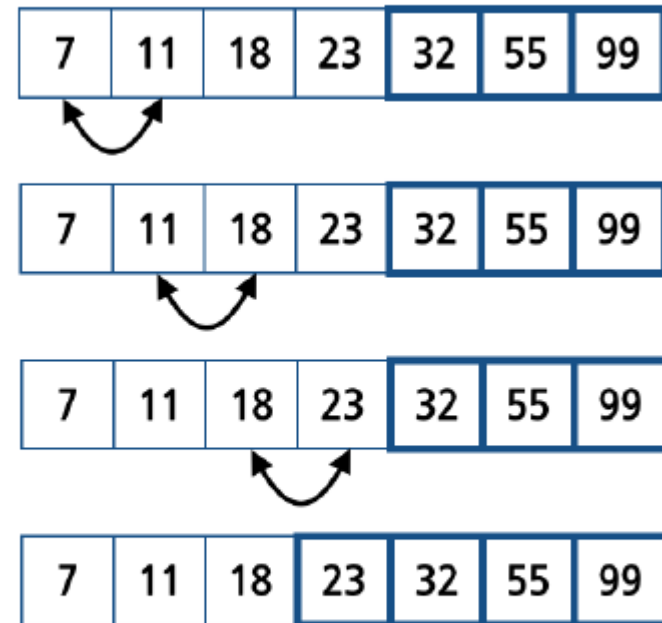


실행 예

3rd loop



4th loop



06. 버블정렬(Bubble sort)



HANYANG
UNIVERSITY

실행 예

5th loop

7	11	18	23	32	55	99
---	----	----	----	----	----	----



7	11	18	23	32	55	99
---	----	----	----	----	----	----



7	11	18	23	32	55	99
---	----	----	----	----	----	----

6th loop

7	11	18	23	32	55	99
---	----	----	----	----	----	----



7	11	18	23	32	55	99
---	----	----	----	----	----	----

Summary

1. 순서열
2. 선택정렬
3. 삽입정렬
4. 합병정렬
5. 퀵정렬
6. 버블정렬

Thanks

Week 6: Recursion and Iteration - Sorting

Instructor: Eunil Park (pa1324@hanyang.ac.kr)

