PROGRAMMING

Lecture 15

Hanbat National University
Dept. of Computer Engineering
Changbeom Choi

OUTLINE

Sorting in Python

Selection sort

Bubble sort

Recursion

SORTING IN PYTHON

Why sort and search ?

25 -50 % of computing time

- Reporting
- Updating
- Inquiring

In most Python implementations, the sorting algorithm called **Timsort**(Tim Peters ,2002) is used, which is a variant of the merge sort(John von Neumann in 1945).

```
>>>L = [4, 2, 3]
>>>L.sort()

>>>print (L.sort())
None
>>>print (L)
[2, 3, 4]
>>>L
[2, 3, 4]
>>>L.sort(reverse = True)
>>>L
[4, 3, 2]
```

L.sort() does change L.

```
>>>L = [4, 2, 3]
>>>sorted(L)
[2, 3, 4]
>>>print (sorted(L))
[2, 3, 4]
>>print (L)
[4, 2, 3]
>>>L
[4, 2, 3]
>>>sorted(L, reverse = True)
[4, 3, 2]
```

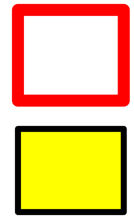
sorted(L) does not change L.

sorted(.) can be used for a sequence

```
>>>T = (4, 2, 3)
>>>D = {“John” : 24, “Mary” : 20, “James” : 22}
>>>sorted(T)
(2, 3, 4)
>>>sorted(D)
[“James”, “John”, “Mary”]
```

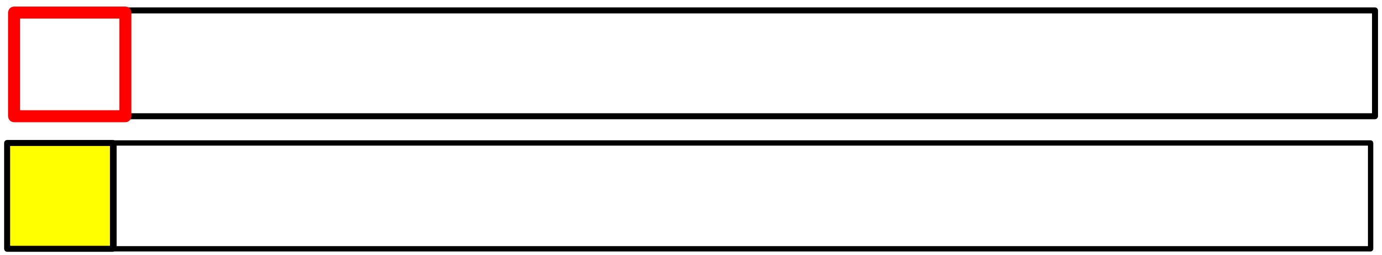
SELECTION SORT

BASIC IDEA

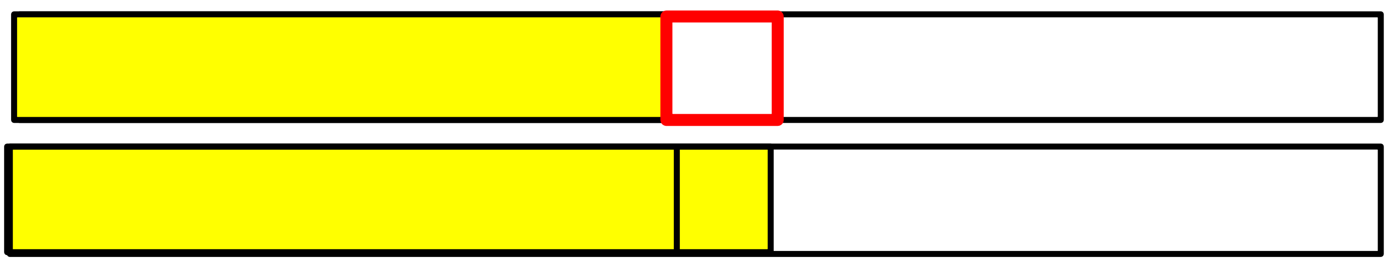

the smallest element

sorted

Step 1



Step k



At the end of the last step

[5, 9, 4, 7, 3] 4 comparisons

[3, 9, 5, 7, 4] 3 comparisons

[3, 4, 9, 7, 5] 2 comparisons

[3, 4, 5, 9, 7] 1 comparison

[3, 4, 5, 7, 9]

[3, 4, 5, 7, 9]

4(4 +1)/2 comparisons

Why ?

How many comparisons in general?

$(n-1)n/\ 2 = n^2/2 - n/2$

$O(\ n^2)$

$1 + 2 + 3 + 4 = 4 (4 + 1) / 2 = 10$ comparisons

In general,

$1 + \ldots\ldots\ldots + (n - 1) = (n - 1)(n - 1 + 1)/2 = (n - 1)n/2$,

where $n = \text{len}(L)$.

Pseudo code

1. Given a list of elements, find the **smallest element** and make it the **first element** of the list. Let the sub-list except the first element be the **remaining list**.

2. Given the remaining list, find the **smallest element** and make it the **first element** of the remaining list. Make a **new remaining list** by excluding the first element.

3. **Repeat Step 2** until the remaining list becomes a **single element**.

```
def selection_sort(L):
    start = 0
    end = len(L)
    while start < end:
        for i in range(start, end):
            if L[i] < L[start]:
                L[start], L[i] = L[i], L[start]
        start += 1
    return L
```

How about start < end -1 ?

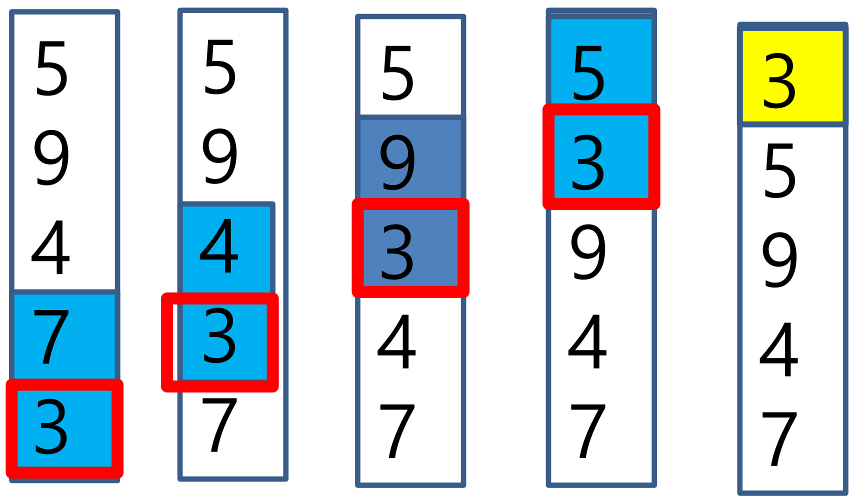
Finding the smallest one in the current sub-list

for the next sub-list

BUBBLE SORT

Basic idea

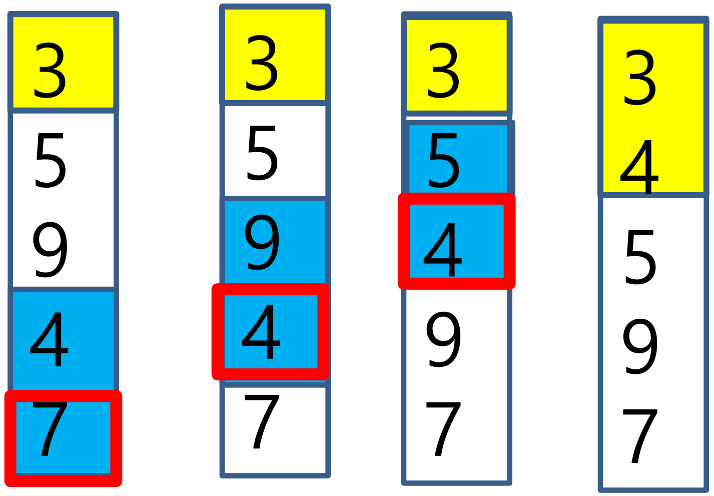
1st round



4 comparisons

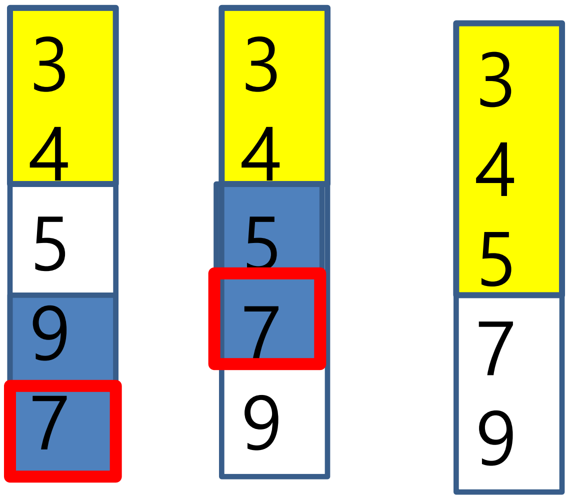
Like a bubble floating up!

2nd round



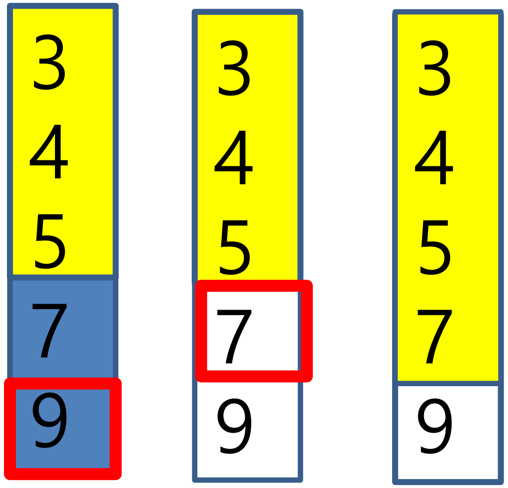
3 comparisons

3rd round



2 comparisons

4th round



1 comparisons

Pseudo code

1. Given a list of elements, let the **smallest element** float up to the top. Let the sub-list except the top element (first element) be the **remaining list**.

2. Given the remaining list, let the **smallest element float up to the top in the remaining list**. Make a **new remaining list** by excluding the top element.

3. **Repeat Step 2** until the remaining list becomes a **single element**.

```
def bubble_sort(L):
    for i in range ( len(L) - 1):
        for j in range(len(L) - 1 - i):
            i1 = (len(L)-1) - (j + 1)
            i2 = (len(L)-1) -  j
            if L[i1] > L[i2]:
                L[i1], L[i2] = L[i2], L[i1]
    return L
```

Sort a sub-list.

Bubbles float up!

RECURSION

Recursion is **defining something** in terms of **itself**. It sounds a bit **odd**, but very **natural** in fact. Here are some examples:

According to the **legal code** of **United States**, a U.S. citizen is

1. Any child born inside United States, or

   the base case

2. Any child born outside United States and at least **one** of whose **parent** is a **U.S. citizen (**satisfying certain condition).

   the recursive(inductive) case

In mathematics, the **factorial function** is defined as follows:

$$f(n) = \begin{cases} 1 \text{ if } n = 1 & \text{base case} \\ n \times f(n-1) \quad \text{if } n > 1 & \text{recursive case} \end{cases}$$

$$\underbrace{f(n)}_{n!} = n \times \underbrace{\boxed{(n-1) \times (n-2) \times \ldots\ldots\ldots \times 2 \times 1}}_{\substack{(n-1)! \\ f(n-1)}}$$

Fibonacci numbers

$$f(n) \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{otherwise} \end{cases}$$

base case

recursive case

n	f(n)
0	1
1	1
2	2
3	3
4	5
5	8
6	13
……	……

Recursive functions

A **function** is said to be **recursive** if the function is defined **in terms of itself**, specifically, the function **calls itself** during its **execution**.

Factorial function

$$f(n) = \begin{cases} 1 \text{ if } n = 1 & \text{base case} \\ n \times f(n-1) \quad \text{if } n > 1 & \text{recursive case} \end{cases}$$

```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n-1)
```

```
def fact(n):
    result = 1
    while n >1:
        result = result * n
        n -= 1
    return result
```

```
def fact(n):
    if n == 1:
        print (1, end= “ ”)
        return 1
    else:
        print  (n, "x ", end=“ ”)
        return n * fact(n-1)

x = fact(7)
print  (“ = “, x)
```

7 x 6 x 5 x 4 x 3 x 2 x | 1 | = 5040

Fibonacci numbers

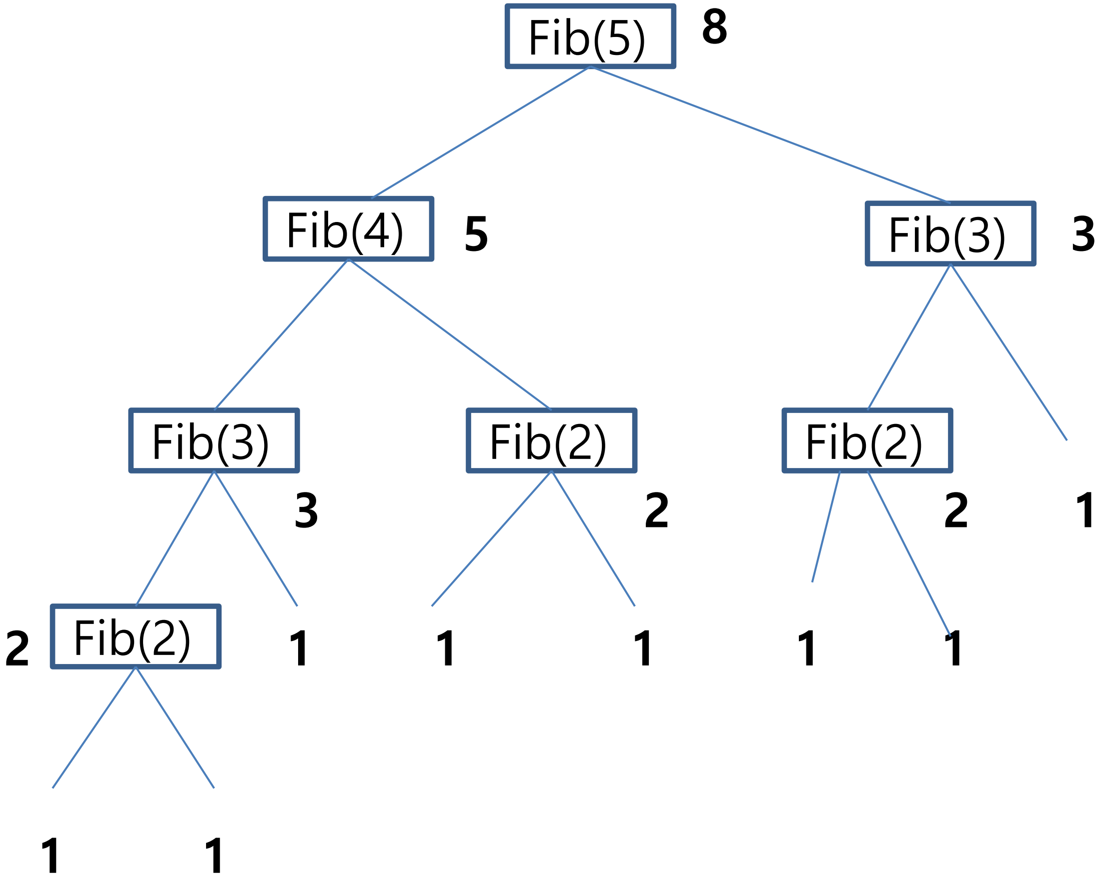
$$f(n) \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) \end{cases}$$

base case

recursive case

```
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n – 1) + fib(n - 2)
```

fib(5)


Fib(5)
8
Fib(4)
5
Fib(3)
3
Fib(3)
3
Fib(2)
2
Fib(2)
2
1
2
Fib(2)
1
1
1
1
1
1
1

```
def fib(n):
    global lv
    lv += 1
    if n == 0 or n == 1:
        print  (" " * 4 * lv, 1)
        lv -= 1
        return 1
    else:
        print (" " * 4 * lv, "fib(", n,")“)
        x = fib(n - 1) + fib(n - 2)
        lv -= 1
        return x

lv = -1
x = fib(5)
print (“₩n“, ”fib(5) = ", x)
```

```
fib( 5 )
    fib( 4 )
        fib( 3 )
            fib( 2 )
                1
                1
            1
        fib( 2 )
            1
            1
    fib( 3 )
        fib( 2 )
            1
            1
        1

fib(5) =  8
```

Palindromes

$$\text{palin(s)} = \begin{cases} \text{True if } n <= 1 & \text{base case} \\ (s[0] == s[-1]) \text{ and palin}(s[1: -1]) \text{ if } n > 1 & \\ & \text{recursive case} \end{cases}$$

```
def is_palin(s):
    if len(s) <= 1:
        return True
    else:
        return (s[0] == s[-1]) and  is_palin(s[1, -1])
```

```python
def is_palin(s):
    global lv
    lv += 1
    if len(s) <= 1:
        display(True)
        lv -= 1
        return True
    else:
        display(s)
        flag = (s[0] == s[-1]) and
                  is_palin(s[1 : -1])
        display(flag)
        lv -= 1
        return flag
```

```
doggod
    oggo
        gg
            True
        True
    True
True
```

```python
def display(s):
    global lv
    if lv == 0:
        print (s)
    else:
        print (" " * 4 * lv,  s)

lv = -1
is_palin( "doggod".lower())
```