

파이썬으로 배우는 데이터 구조



한동대학교
전산전자공학부
김영섭 교수



학습 목표

파이썬의 `sort()`, `sorted()`의 차이를 알아보고
버블(Bubble) 정렬의 알고리즘을 이해할 수 있다

Data Structures in Python

Chapter 5 - 1

- Binary Search
- Recursive Binary Search
- **Bubble sort**
- Selection sort
- Insertion sort

Agenda & Readings

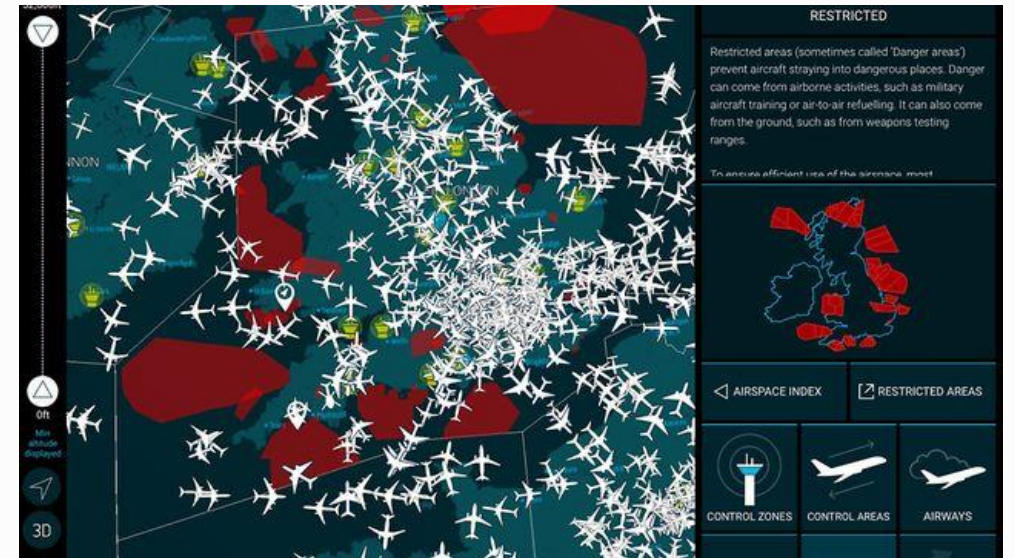
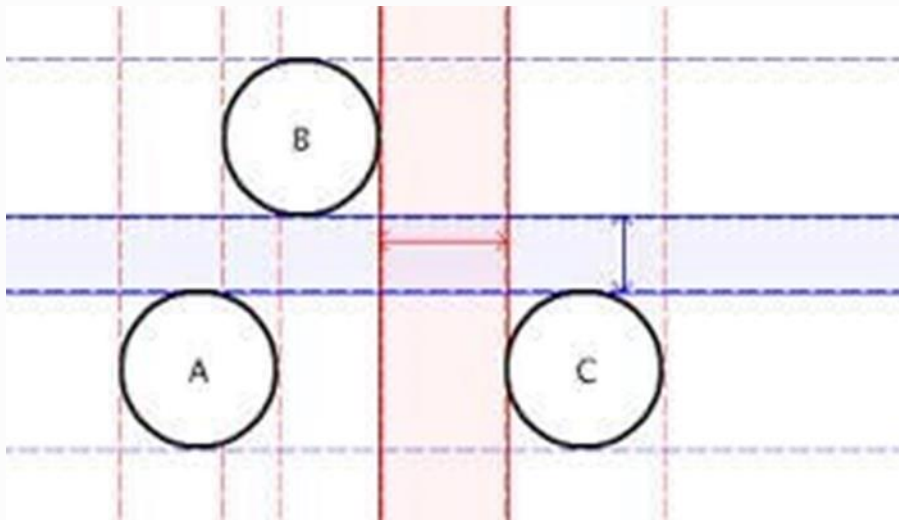
- Agenda
 - Motivation
 - Python sorted() function and sort() method
 - Bubble sort algorithm
 - Time complexity - Big O
- Reference:
 - Problem Solving with Algorithms and Data Structures
 - Chapter 5 Search, Sorting and Hashing

Sorting: One of the Most Common Activities on Computers

- **Example 1:**
 - **Alphabetically sorted names:**
 - e.g., names in phone book, street names in map, file names in a folder
 - **Advantages:**
 - Can use efficient search algorithms:
 - Binary search finds item in $O(\log n)$ time
- **Example 2:**
 - **Sorted numbers:**
 - e.g., house prices, student IDs, grades, rankings
 - **Advantages:**
 - Can use efficient search algorithms (see example 1)
 - Easy to find position or range of values in sorted list, e.g., minimum value, median value, quartile values, all students with A grades, all houses within a certain price range etc.

Sorting: One of the Most Common Activities on Computers

- **Example 3:**
 - Sort objects in space.
 - e.g., Objects in a street, Objects in space
 - Advantages:
 - Can use efficient search algorithms, e.g., for collision detection



Sorting: Important Properties to Investigate

- **How efficient** is the sorting algorithm?
 - Note: can depend on order of input data set, e.g., is it almost sorted or completely unsorted?
- **How much memory** does sort algorithm require?
- **How easy** is algorithm to implement?
 - for simple problems and small data sets, simple sorting algorithm usually sufficient

Sorting: Need a comparison operator

- Any information which needs to be kept in sorted order will involve the **comparison** of items ($<$, $=$, $>$), e.g., strings and numbers:
 - ints/floats
 - $-34 < -1 < 0 < 1 < 245$
 - Characters
 - $A < \dots < \dots < Y < Z < a < b < c \dots < y < z$
 - Strings
 - $'Hungry' < 'Money' < 'More' < 'money' < 'work'$
- Any information which needs to be kept in sorted order will have a key, the sort **key** (e.g., id, name, code number, ...).
 - The key determines the position of the individual object in the collection.
 - Commonly the key is a number.
 - When comparing keys which are strings, the Unicode (ASCII) values of the string are used (e.g., 'a' is 0x000061, 'A' is 0x000041 and ' ' is 0x000020).

Python sorted() function – 1

- Python has an inbuilt sort function: `sorted()`
 - The `sorted()` function takes any iterable and returns a list containing the sorted elements. (Note that all sequences are iterable.)

```
a = [5, 2, 3, 1, 4]
b = sorted(a)
print("a: ", a)
print("b: ", b)
print(b == a)
```

```
a: [5, 2, 3, 1, 4]
b: [1, 2, 3, 4, 5]
False
```

Python sorted() function – 2

- Python has an inbuilt sort function: `sorted()`
 - The `sorted()` function takes any iterable and returns a list containing the sorted elements. (Note that all sequences are iterable.)
 - Example: List

```
a = [5, 2, 3, 1, 4]
b = sorted(a)
print("a: ", a)
print("b: ", b)
print(b == a)
```

```
a: [5, 2, 3, 1, 4]
b: [1, 2, 3, 4, 5]
False
```

- Example: String

```
a = "think"
b = sorted(a) # sorted always returns a list
print("a:", a)
print("b:", b)
print(b == a)
```

```
a: think
b: ['h', 'i', 'k', 'n', 't']
False
```

Python sorted() function – 2

- Python has an inbuilt sort function: `sorted()`
 - Example: Dict
 - For dictionary, `sorted()` returns **sorted list of keys**, and sorts output by **keys**.

```
a = {4:5, 2:9, 1:6, 3:7}
```

```
b = sorted(a)
```

```
print("a: ", a)
```

```
print("b: ", b)
```

```
print(b == a)
```

```
a: {1: 6, 2: 9, 3: 7, 4: 5}
```

```
b: [1, 2, 3, 4]
```

```
False
```

```
a = {'kiwi':5, 'banana':9, 'apple':6, 'orange':7}
```

```
b = sorted(a)
```

```
print("a: ", a)
```

```
print("b: ", b)
```

```
print(b == a)
```

```
a: {'kiwi': 5, 'banana': 9, 'apple': 6, 'orange': 7}
```

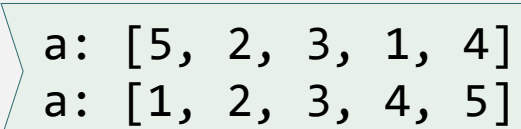
```
b: ['apple', 'banana', 'kiwi', 'orange']
```

```
False
```

Python list method, sort()

- As well as the Python built-in `sorted()` **function**, the `sort()` **method** can be used to sort the elements of a list **in place**.

```
a = [5, 2, 3, 1, 4]
print("a: ", a)
a.sort()
print("a: ", a)
```



a: [5, 2, 3, 1, 4]
a: [1, 2, 3, 4, 5]

Python `sorted()` function, `list sort()` method

- We already have the Python sorting functions.
Why bother looking at sorting algorithms?
 - It gives us a greater understanding of how our programs work.
 - Best sorting function depends on application.
 - Useful for developing sorting algorithms for specific applications
- In particular, we are interested in how much processing it takes to sort a collection of items (i.e., the Big O).
 - Also, as Wikipedia says: "useful new algorithms are still being invented, with the now widely used Timsort dating to 2002, and the library sort being first published in 2006."
 - In Python, Timsort is used (for both `sorted()` and `sort()`).

Sorting: The Expensive Bits

- In order to sort items, we will need to **compare** items and swap them if they are out of order.
- Number of **comparisons** and the number of **swaps** are the costly operations in the sorting process, and these affect the efficiency of a sorting algorithm.

Sorting Considerations

- An **internal sort** requires that the collection of data fit entirely in the computer's main memory.
- An **external sort**: the collection of data will not fit in the computer's main memory all at once but must reside in secondary storage.
- For very large collections of data it is costly to create a new structure (list) and fill it with the sorted elements so we will look at sorting **in place**.
- **One pass** is defined as one trip through the data structure (or part of the structure) comparing and, if necessary, swapping elements along the way. (In these examples the data structure is a list of ints.)
- In these discussions we sort from smallest (on the left of the list) to largest (on the right of the list).

Bubble Sort

- IDEA:
 - Given is a list L of n value $\{L[0], \dots, L[n-1]\}$
 - Divide list into unsorted (left) and sorted part (right - initially empty):
Unsorted: $\{L[0], \dots, L[n-1]\}$ **Sorted:** $\{\}$
 - In each pass compare adjacent elements and swap elements not in correct order
→ largest element is “bubbled” to the right of the unsorted part
 - Reduce size of unsorted part by one and increase size of sorted part by one.
After i-th pass:
Unsorted: $\{L[0], \dots, L[n-1-i]\}$ **Sorted:** $\{L[n-i], \dots, L[n-1]\}$
 - Repeat until unsorted part has a size of 1 - then all elements are sorted

Bubble Sort Algorithm

- Given is a list L of n value $\{L[0], \dots, L[n-1]\}$
 - Divide list into unsorted (left) and sorted part (right - initially empty):
Unsorted: $\{L[0], \dots, L[n-1]\}$ **Sorted:** $\{\}$
 - In each pass compare **adjacent elements** and swap elements not in correct order
→ largest element is “bubbled” to the right of the unsorted part
 - Reduce size of unsorted part by one** and increase size of sorted part by one.
After i-th pass:
Unsorted: $\{L[0], \dots, L[n-1-i]\}$ **Sorted:** $\{L[n-i], \dots, L[n-1]\}$
 - Repeat until unsorted part has a size of 1, then all elements are sorted

29	10	14	37	13
10	14	29	13	37
10	14	13	29	37
10	13	14	29	37
10	13	14	29	37

List to sort

PASS 1 (4 Comp, 3 Swap)

PASS 2 (3 Comp, 1 Swap)

PASS 3 (2 Comp, 1 Swap)

PASS 4 (1 Comp, 0 Swap)

Bubble Sort – Exercise

54	26	93	17	77	31	44	55	20
26	54	17	77	31	44	55	20	93
27	17	54	31	44	55	20	77	
17	26	31	44	54	20	55		
17	26	31	44	20	54			
17	26	31	20	44				
17	26	20	31					
17	20	26						
17	20							

List to sort

- PASS 1 (8 Comp, 7 Swap)
- PASS 2 (7 Comp, 5 Swap)
- PASS 3 (6 Comp, 4 Swap)
- PASS 4 (5 Comp, 1 Swap)
- PASS 5 (4 Comp, 1 Swap)
- PASS 6 (3 Comp, 1 Swap)
- PASS 7 (2 Comp, 1 Swap)
- PASS 8 (1 Comp, 0 Swap)

Some Useful Python Features - print_part(), swap()

```
def print_part(a, i, j):  
    print(i, j, a[i:j])
```

```
a = [54, 26, 93, 17, 77, 31, 44, 55, 20]  
for x in range(0, len(a), 3):  
    print(a[x], end=" ")  
print_part(a, 2, 5)  
print_part(a, 0, 9)
```

```
54 17 44  
2 5 [93, 17, 77]  
0 9 [54, 26, 93, 17, 77, 31, 44, 55, 20]
```

```
def swap(a, i, j):  
    temp = a[i]  
    a[i] = a[j]  
    a[j] = temp
```



```
def swap(a, i, j):  
    a[i], a[j] = a[j], a[i]
```

Bubble Sort Code

- Code

```
def swap(a, i, j):  
    a[i], a[j] = a[j], a[i]  
  
def bubble_sort(a):  
    for pass in range(len(a)-1, 0, -1):  
        for i in range(0, pass):  
            if a[i] > a[i+1]:  
                swap(a, i, i+1)  
                #print(pass, "-", a)  
  
if __name__ == '__main__':  
    a = [54, 26, 93, 17, 77, 31, 44, 55, 20]  
    print("before: ", a)  
    bubble_sort(a)  
    print(" after: ", a)
```

```
before:  [54, 26, 93, 17, 77, 31, 44, 55, 20]  
after:   [17, 20, 26, 31, 44, 54, 55, 77, 93]
```

Bubble Sort - Big O

- For a list with n elements:

- The number of comparisons?

pass 1	pass 2	pass 3	...	last pass
$n-1$	$n-2$	$n-3$...	1

$$1 + 2 + \dots + (n-3) + (n-2) + (n-1) = \frac{1}{2}(n^2 - n)$$

- Big O of the bubble sort is $O(n^2)$
 - The number of data increases 10 times, then it takes a 100 times longer.
- On average, the number of swaps is half the number of comparisons.

Bubble Sort - Summary

- Sorting is a necessary tool in computing.
- The bubble sort algorithm is simple, but slow.
 - It performs lots of comparisons ($O(n^2)$) and many swaps in each pass additionally.

학습 정리

- 1) 정렬의 성능은 비교와 교환 작업의 횟수로 결정된다
- 2) 버블 정렬 알고리즘은 단순하지만 속도는 느리다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다

