파이썬으로 배우는 데이터 구조

한동대학교
전산전자공학부
김영섭 교수

학습 목표

퀵(Quick) 정렬의 알고리즘을
이해하고 구현할 수 있다

Data Structures in Python
Chapter 5 - 2

- Merge sort
- **Quick sort Algorithm**
- Quick sort Analysis
- Empirical Analysis

Agenda & Readings

- Agenda
  - Quick sort algorithm
    - $n \log(n)$ algorithm,
    - Divide and conquer algorithm
- Reference:
  - Problem Solving with Algorithms and Data Structures
    - Chapter 5 Search, Sorting and Hashing: Quick sort
    - Wikipedia Quick sort
    - [알고리즘] 퀵정렬

Quick sort

- Quick sort invented by British computer scientist C.A.R. Hoare in 1960
- Quick sort is another divide and conquer algorithm.
- Time complexity is $O(n \log n)$ average, $O(n^2)$ worst case, faster than merge sort in general.

Quick sort - Algorithm

- Choose a pivot element from the array. Although we can choose any element in the array as a pivot, it's easy to implement if we choose the rightmost element of the subarray.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**

pivot

- Partition or reorder the array so that all values smaller than the pivot are moved before it and all values larger (or equal) than the pivot are moved after it.

32	23	43	39	57	16	92	81	75	**65**

pivot

  - When this is done, the pivot is in its final position.

32	23	43	39	57	16	**65**	81	75	92

sorted

- Conquer by repeating steps above for the left side and right side of the pivot recursively, except the pivot itself since it is sorted and positioned at the right place,

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2	j=4 i=2	j=5 i=3	j=6 i=4	j=7 i=5	j=8 i=5	pivot

j=0,i=-1

while **j** traverses from **low** to **hi-1**
i increments only when **a[j] < pivot**

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2	j=4 i=2	j=5 i=3	j=6 i=4	j=7 i=5	j=8 i=5	pivot

j=0,i=-1

Now, find elements to swap.

while **j** traverses from **low** to **hi-1**
i increments only when **a[j] < pivot**

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2	j=4 i=2	j=5 i=3	j=6 i=4	j=7 i=5	j=8 i=5	pivot
i=-1				swap						
swap(2,3)	32	23	43	**81**	92	39	57	16	75	**65**

While j increments,
if a[j] < pivot, increments i and
if i != j then swap a[i] and a[j].

while j traverses from low to hi-1
i increments only when a[j] < pivot

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2 swap	j=4 i=2	j=5 i=3 swap	j=6 i=4	j=7 i=5	j=8 i=5	pivot
swap(2,3)	32	23	43	**81**	92	39	57	16	75	**65**
swap(3,5)	32	23	43	39	92	**81**	57	16	75	**65**

While j increments,
if a[j] < pivot, increments i and
if i != j then swap a[i] and a[j].

while j traverses from low to hi-1
i increments only when a[j] < pivot

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted.

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2 swap	j=4 i=2	j=5 i=3 swap	j=6 i=4 swap	j=7 i=5	j=8 i=5	pivot
swap(2,3)	32	23	43	**81**	92	39	57	16	75	**65**
swap(3,5)	32	23	43	39	92	**81**	57	16	75	**65**
swap(4,6)	32	23	43	39	57	**81**	**92**	16	75	**65**

while **j** traverses from **low** to **hi-1**
i increments only when **a[j] < pivot**

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2 swap	j=4 i=2	j=5 i=3 swap	j=6 i=4 swap	j=7 i=5 swap	j=8 i=5	pivot
swap(2,3)	32	23	43	**81**	92	39	57	16	75	**65**
swap(3,5)	32	23	43	39	92	**81**	57	16	75	**65**
swap(4,6)	32	23	43	39	57	**81**	**92**	16	75	**65**
swap(5,7)	32	23	43	39	57	16	**92**	**81**	75	**65**

not sorted

while **j** traverses from **low** to **hi-1**
i increments only when **a[j] < pivot**

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

i+1 becomes the pivot index for element sorted

Quick sort - Partition

- Choose a pivot element and partition the array in place such that all elements to the left of the pivot element are smaller, while all elements to the right are greater than the pivot.

index	0	1	2	3	4	5	6	7	8	9
value	32	23	81	43	92	39	57	16	75	**65**
j=lo i=lo-1	j=0 i=0	j=1 i=1	j=2 i=1	j=3 i=2 swap	j=4 i=2	j=5 i=3 swap	j=6 i=4 swap	j=7 i=5 swap	j=8 i=5	pivot
swap(2,3)	32	23	43	**81**	92	39	57	16	75	**65**
swap(3,5)	32	23	43	39	92	**81**	57	16	75	**65**
swap(4,6)	32	23	43	39	57	**81**	**92**	16	75	**65**
swap(5,7)	32	23	43	39	57	16	**92**	**81**	75	**65**
swap(6,9) swap(i+1,hi)	32	23	43	39	57	16	**65**	**81**	75	**92**

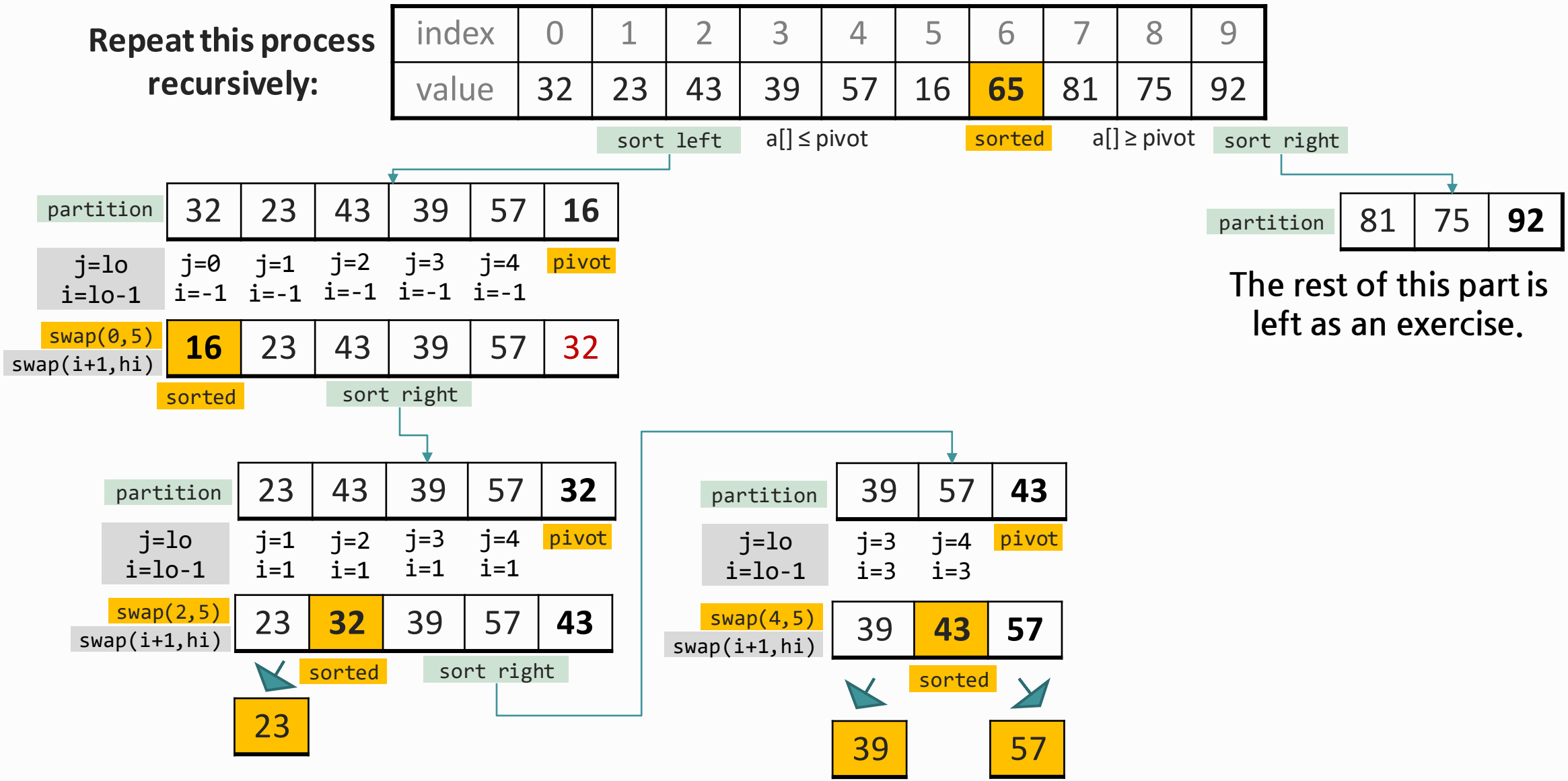
recursively | sort left | a[] ≤ pivot | sorted | a[] ≥ pivot | sort right

```
while j traverses from low to hi-1
i increments only when a[j] < pivot
```

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

Quick sort - Partition

Repeat this process recursively:



Quick sort

Repeat this process recursively:

index	0	1	2	3	4	5	6	7	8	9
value	32	23	43	39	57	16	**65**	81	75	92

sort left — a[] ≤ pivot — sorted — a[] ≥ pivot — sort right

partition: 32 | 23 | 43 | 39 | 57 | **16**

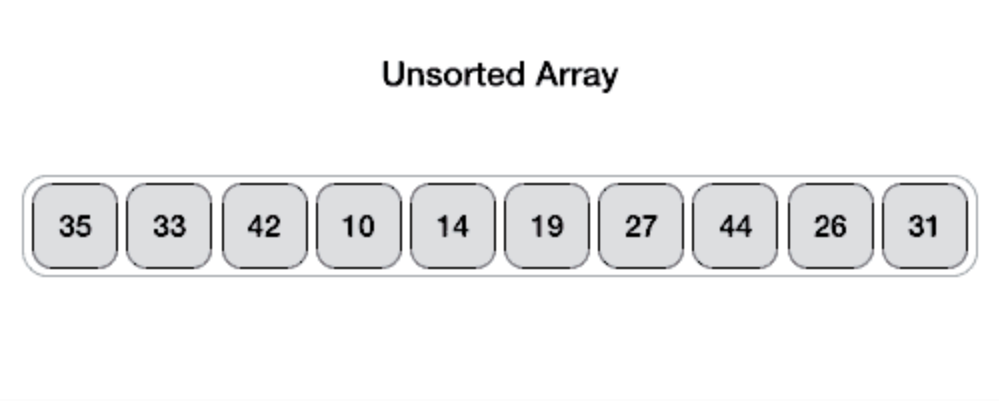
partition: 81 | 75 | **92**

```
def qsort(a, lo, hi):
    if lo >= hi: return

    pi = partition(a, lo, hi)      partition
    qsort(a, lo, pi - 1)           sort left
    qsort(a, pi + 1, hi)           sort right

def sort(a):
    qsort(a, 0, len(a) - 1)        quick sort
```

```
def partition(a, lo, hi):
    pivot = a[hi]
    i = lo - 1;
    j = lo
    while j <= hi - 1:                    scan
        if a[j] < pivot:
            i += 1
            if i != j:                    swap
                a[j], a[i] = a[i], a[j]
        j += 1
    a[hi], a[i+1] = a[i+1], a[hi]         sorted
    return i + 1
```

Summary

- Quick sort is one of the most efficient sorting algorithms.
- It is based on the splitting of an array (**partition**) into smaller ones and rearrange based on the comparison with '**pivot**' element selected.
- Like merge sort, quick sort also falls into the category of **divide and conquer** approach of problem-solving methodology.



학습 정리

1) 퀵 정렬은 가장 효과적인 정렬 알고리즘 중 하나이다

2) 퀵 정렬의 핵심은 피봇(pivot)으로 설정된 요소를 중심으로 배열을 구분(partition)하여 재귀함수를 적용한다

3) 병합 정렬과 퀵 정렬은 분할 정복법(divide and conquer)을 사용하는 알고리즘이다

파이썬으로 배우는 데이터 구조

수고했습니다
곧 다음 시간에
다시 뵙겠습니다