sorting

Quick sort

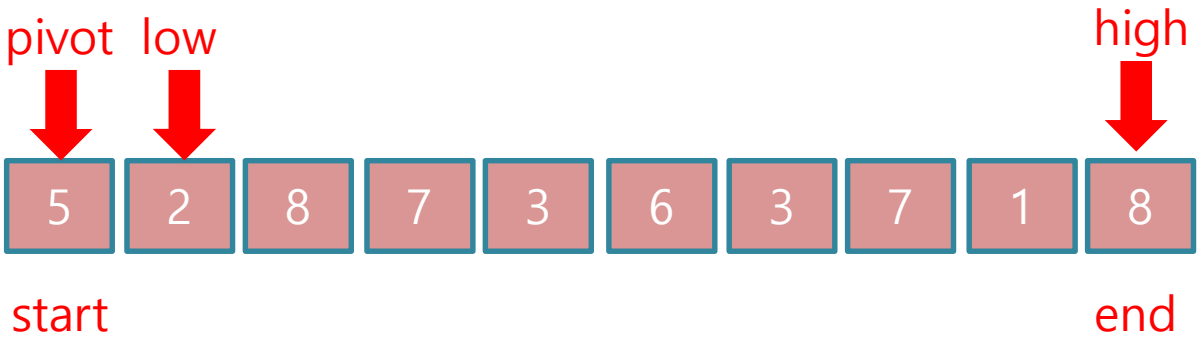
정렬 알고리즘
: Divide and Conquer

분할정복기법
- 문제를 분할하여 해결
- 재귀 함수를 이용

```
def quick_sort(data, start, end):
    #탈출조건
    if start >= end:
        return

    idx_pivot = partition(data, start, end)

    quick_sort(data, start, idx_pivot - 1)
    quick_sort(data, idx_pivot + 1, end)
```

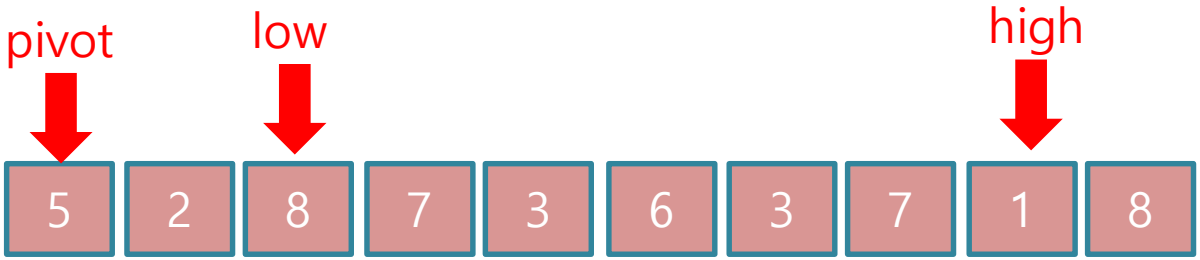
정렬 알고리즘
: Quicksort

Pivot을 맨 왼쪽에!
Low는 그 다음에!



정렬 알고리즘
: Quicksort

Low의 값이 pivot 값보다 클 때까지 오른쪽으로 이동

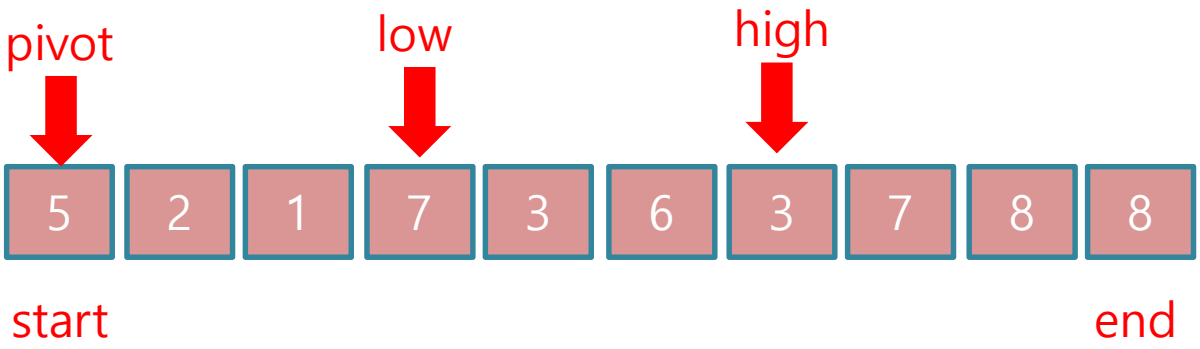


high의 값이 pivot 값보다 작을 때까지 왼쪽으로 이동

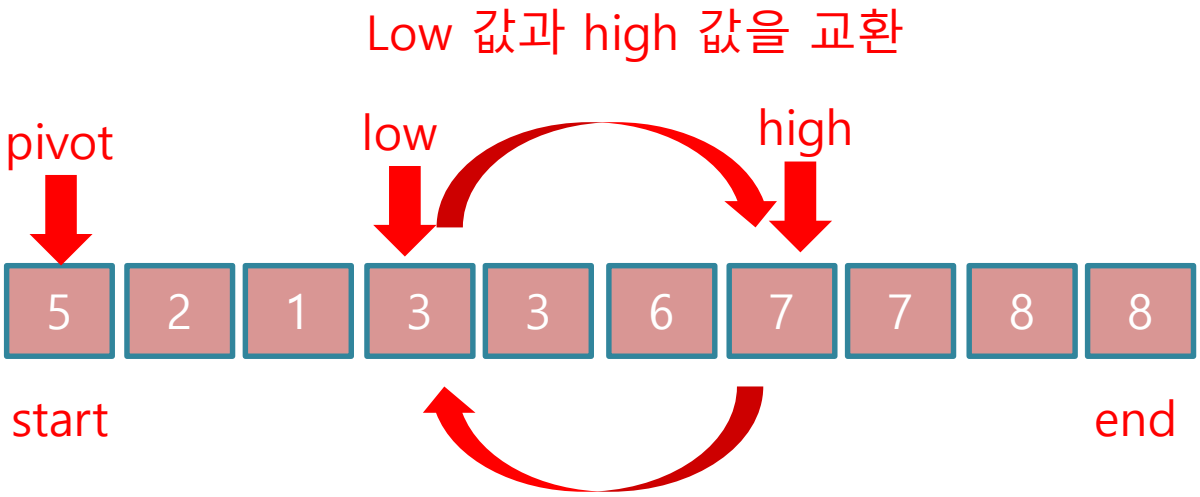
정렬 알고리즘
: Quicksort

정렬 알고리즘
: Quicksort

Low와 high가 교차할 때 까지 계속!

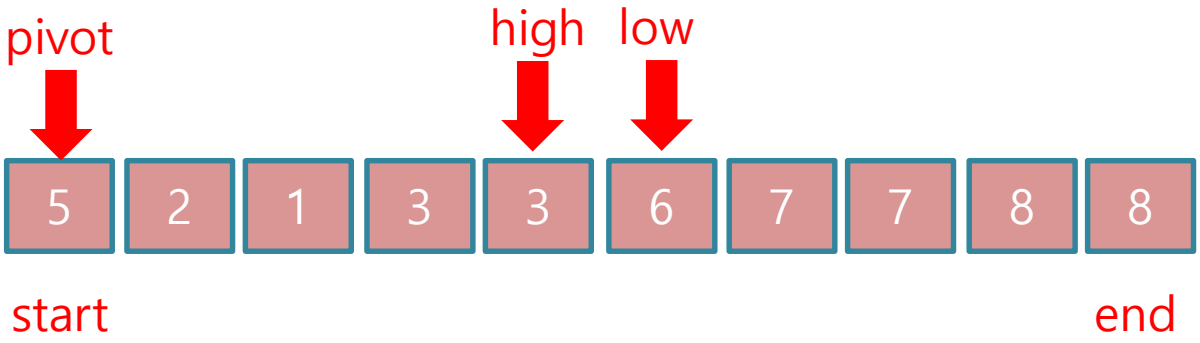


정렬 알고리즘
: Quicksort



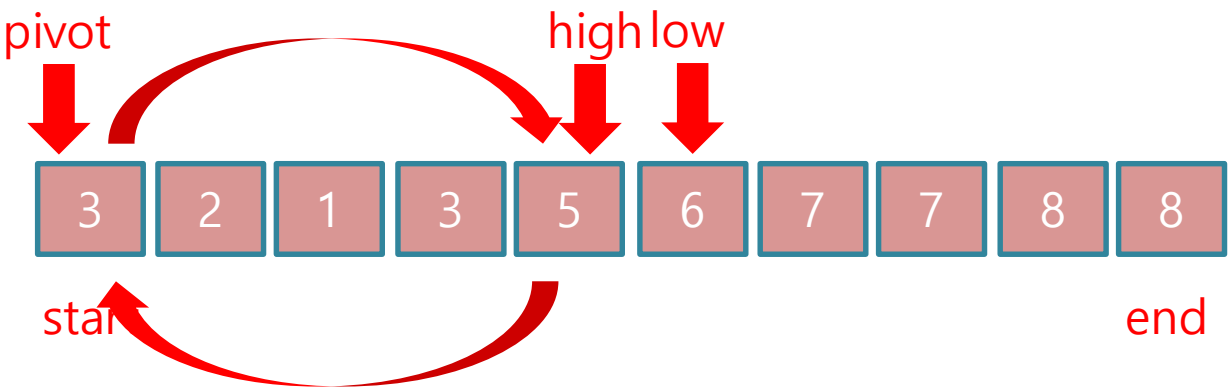
정렬 알고리즘
: Quicksort

Low와 high가 교차할 때!!!!!



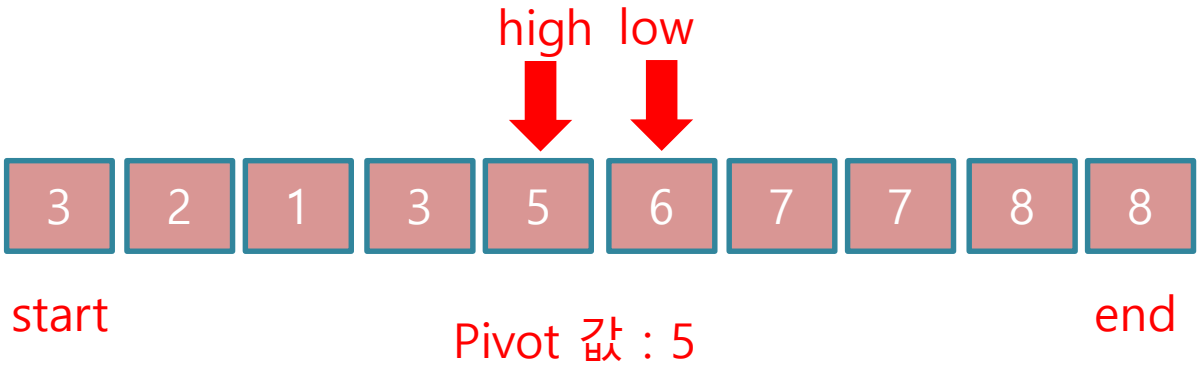
정렬 알고리즘
: Quicksort

Pivot과 high를 교환



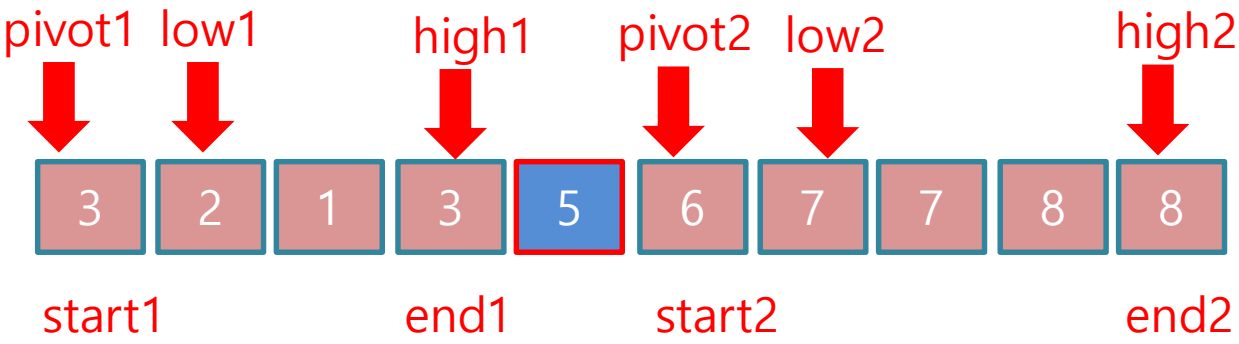
정렬 알고리즘
: Quicksort

Pivot 값을 기준으로 왼쪽에는 작은 값만
오른쪽에는 큰 값만 모였다!!



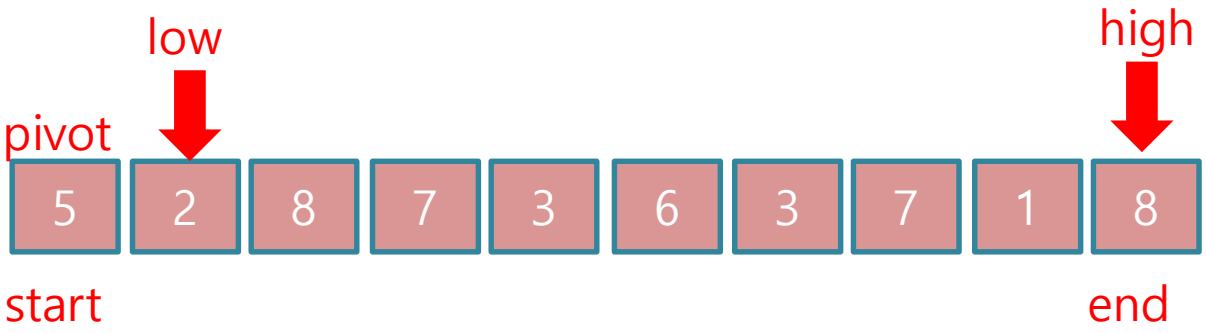
정렬 알고리즘
: Quicksort

Pivot 은 제외 하고 pivot의 왼쪽과 오른쪽에서 같은 알고리즘을 수행



정렬 알고리즘
: Quicksort

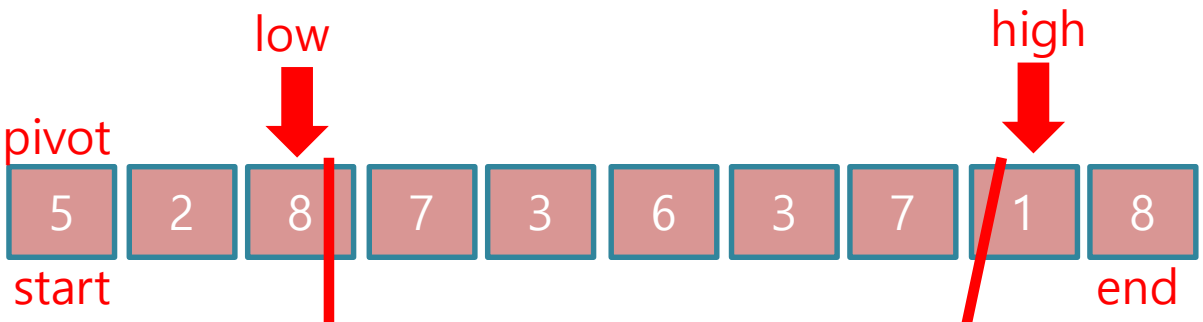
```
def partition(data, start, end):
```




```
pivot = data[start]

low = start + 1
high = end
```

정렬 알고리즘
: Quicksort

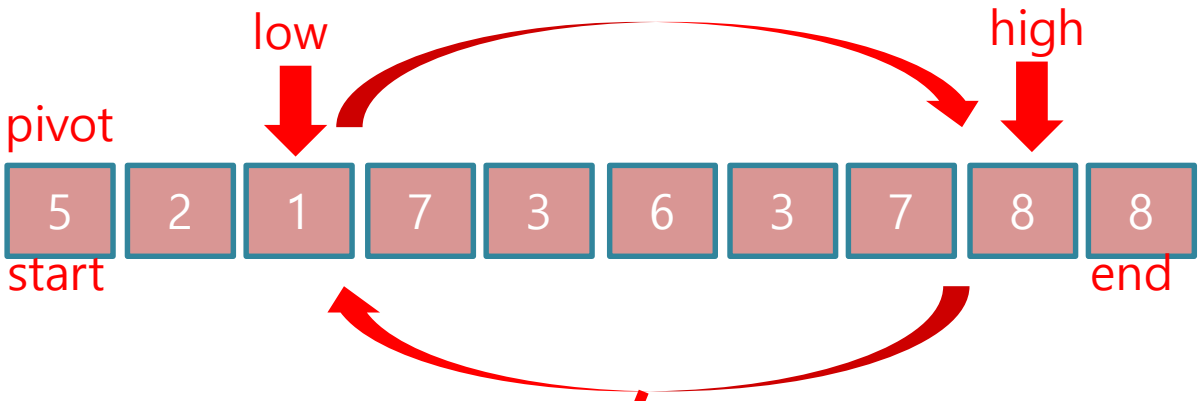
```
def partition(data, start, end):
```




```
while low <=high:
    while low <=end and data[low] <= pivot:
        low += 1

    while high >=(start + 1) and data[high] >= pivot:
        high -= 1
```

정렬 알고리즘
: Quicksort

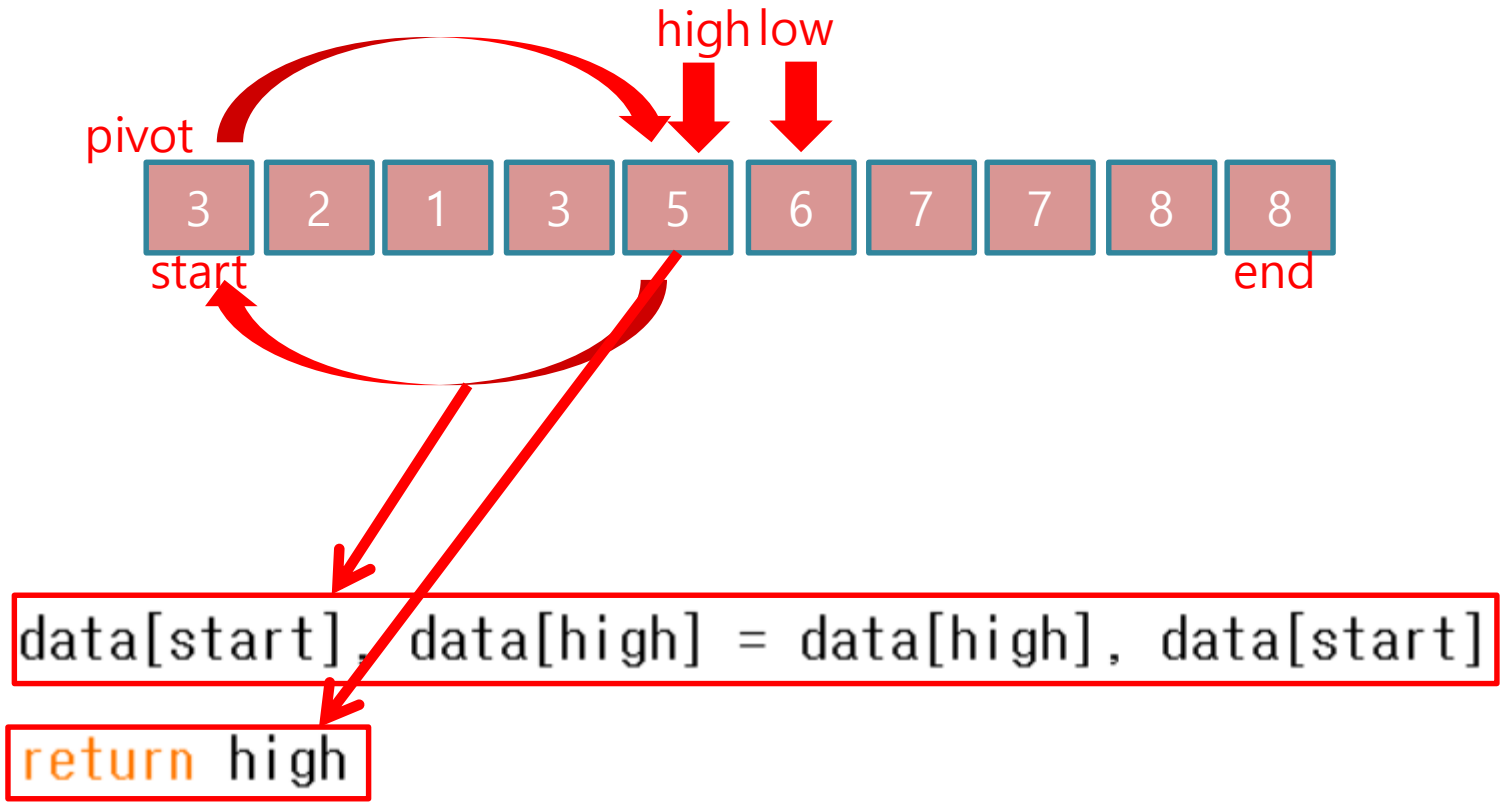
```
def partition(data, start, end):
```




```
    while low <=high:
        while low <=end and data[low] <= pivot:
            low += 1

        while high >=(start + 1) and data[high] >= pivot:
            high -= 1

        if low <= high:
            data[low], data[high] = data[high], data[low]
```

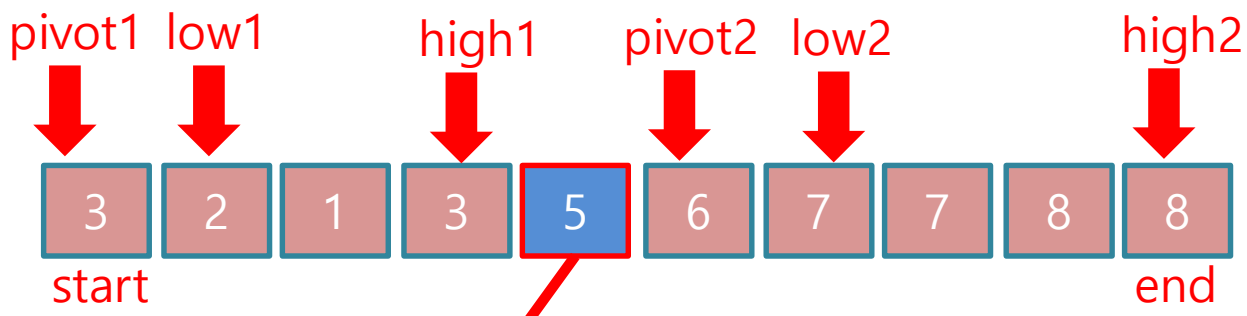
정렬 알고리즘
: Quicksort

```
def partition(data, start, end):
```



정렬 알고리즘
: Quicksort

```
def quick_sort(data, start, end):
```




```
def quick_sort(data, start, end):
    #탈출조건
    if start >= end:
        return

    idx_pivot = partition(data, start, end)

    quick_sort(data, start, idx_pivot - 1)
    quick_sort(data, idx_pivot + 1, end)
```

정렬 알고리즘
: Quicksort

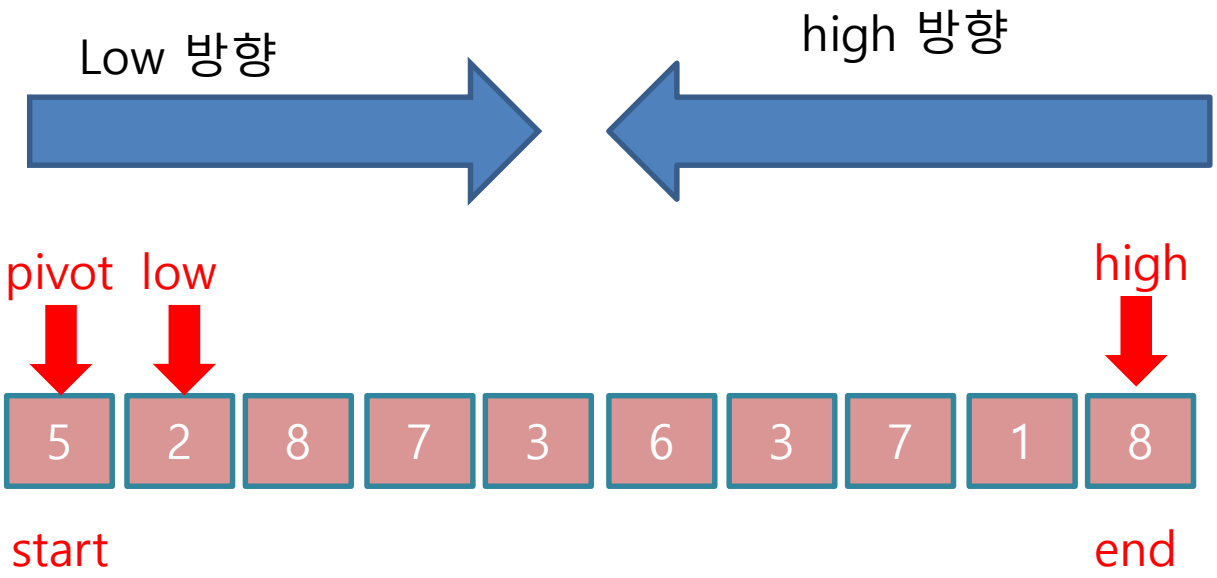
빅오를 계산해봅시다

연산 횟수를 계산하기 위해 주요 비교 연산 부분을 결정합니다.

Low와 pivot, high와 pivot과의 크기 비교 연산!!

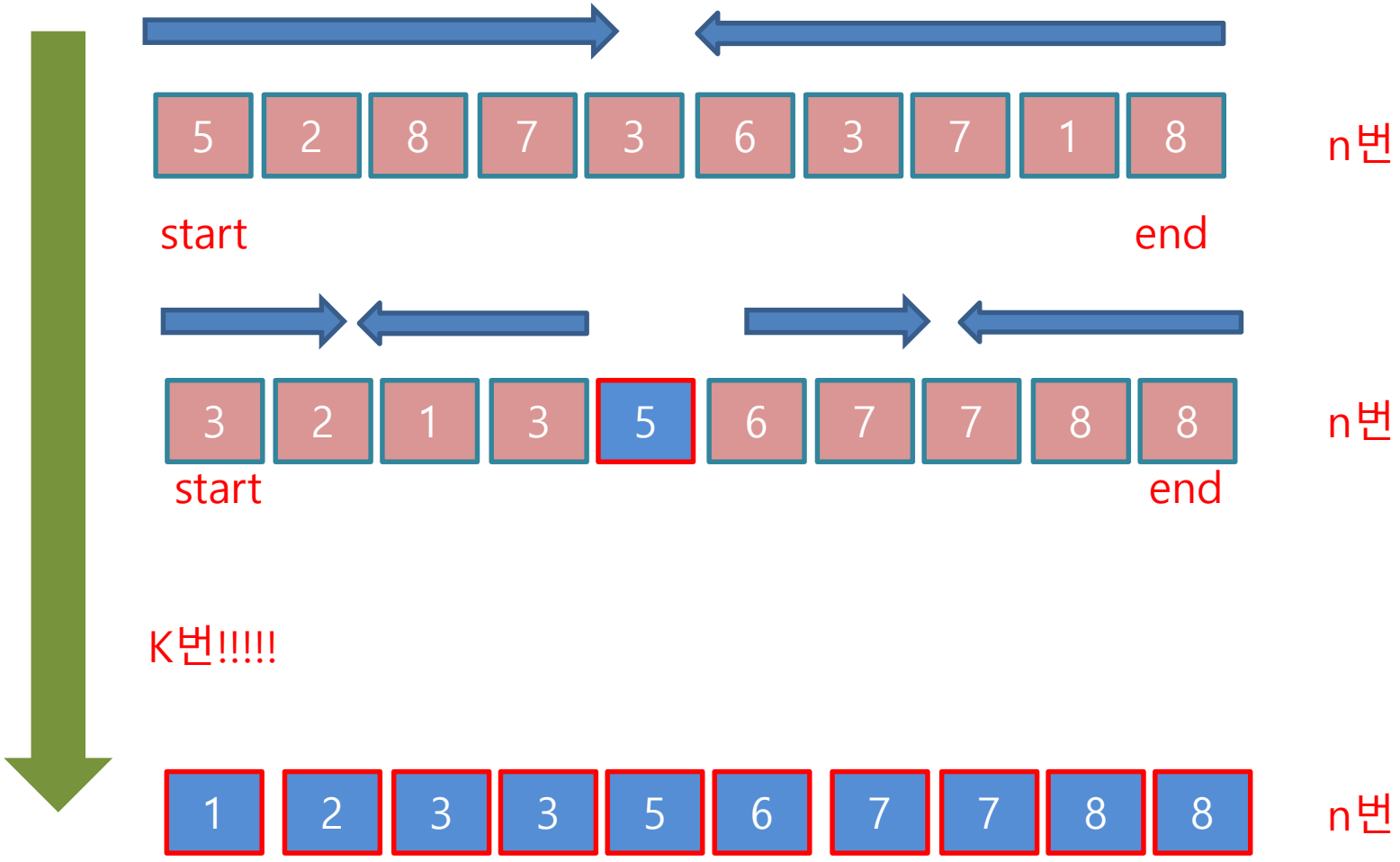
```
while low <=high:
    while low <=end and data[low] <= pivot:
        low += 1

    while high >=(start + 1) and data[high] >= pivot:
        high -= 1
```

정렬 알고리즘
: Quicksort



즉 데이터 개수와 같이 n번 연산합니다

정렬 알고리즘
: Quicksort



$$T(n) = n * k$$

정렬 알고리즘
: Quicksort

$$n * \left(\frac{1}{2}\right)^k = 1 \quad \Longrightarrow \quad k = \log_2 n$$

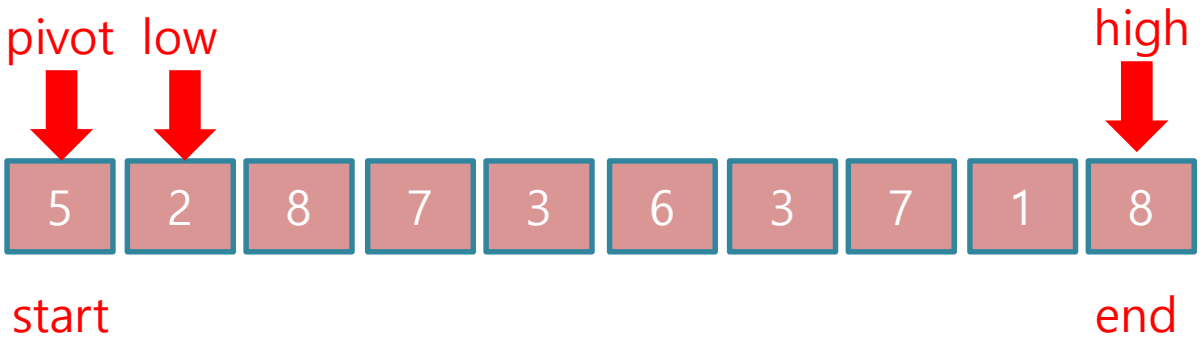
$$T(n) = n * \log_2 n$$

$$O(n * log\, n)$$

정렬 알고리즘
: Quicksort

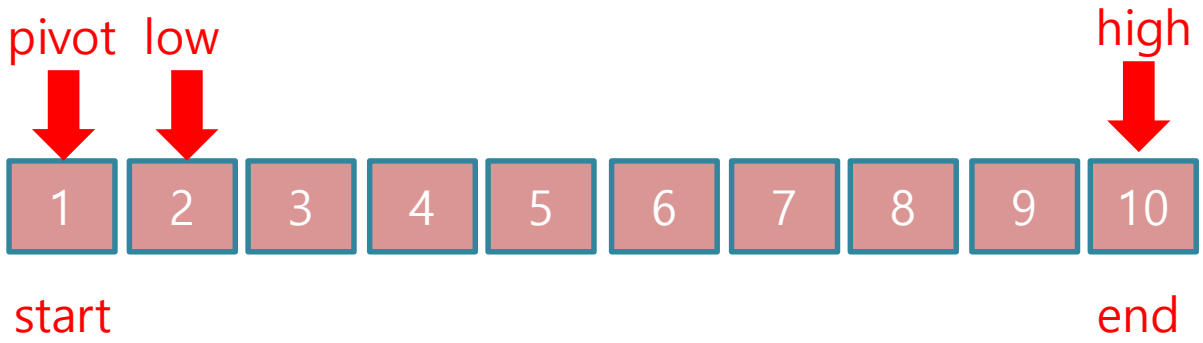
항상 pivot을 맨 왼쪽에 두면.....

Pivot을 맨 왼쪽에!
Low는 그 다음에!



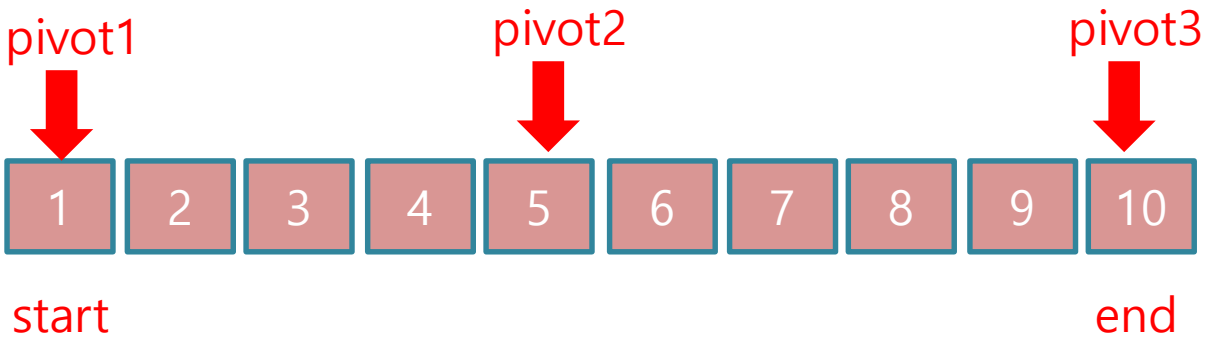
정렬 알고리즘
: Quicksort

이렇게 완벽하게 정렬된 경우 배열이 두 개로 쪼개지지 않으므로
성능이 좋지 않습니다



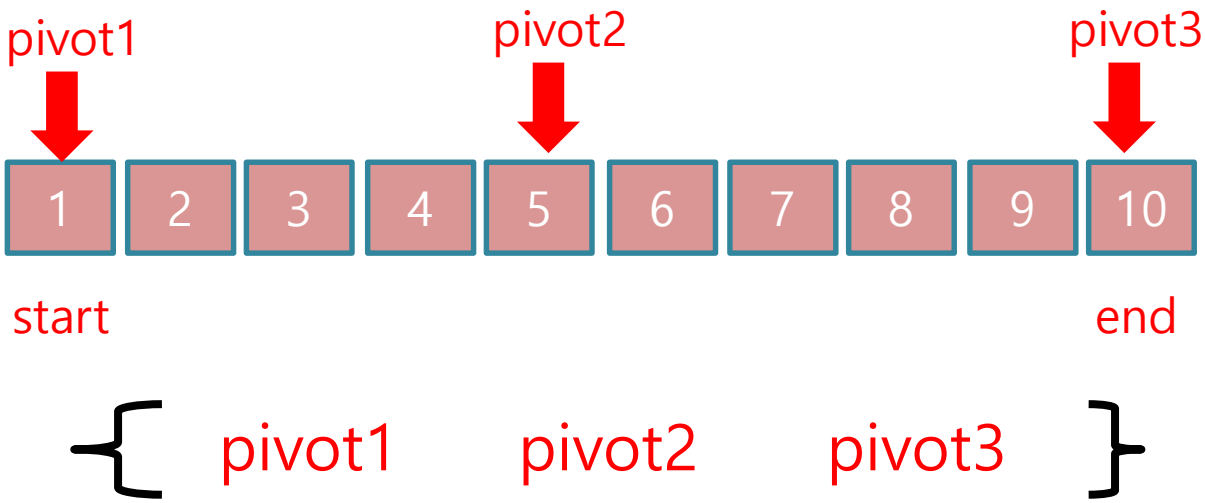
정렬 알고리즘
: Quicksort

이럴 경우에는 pivot을 결정할 때
맨 왼쪽, 가운데, 맨 오른쪽 값을 비교해 중간 값을
Pivot으로 만듭니다



정렬 알고리즘
: Quicksort

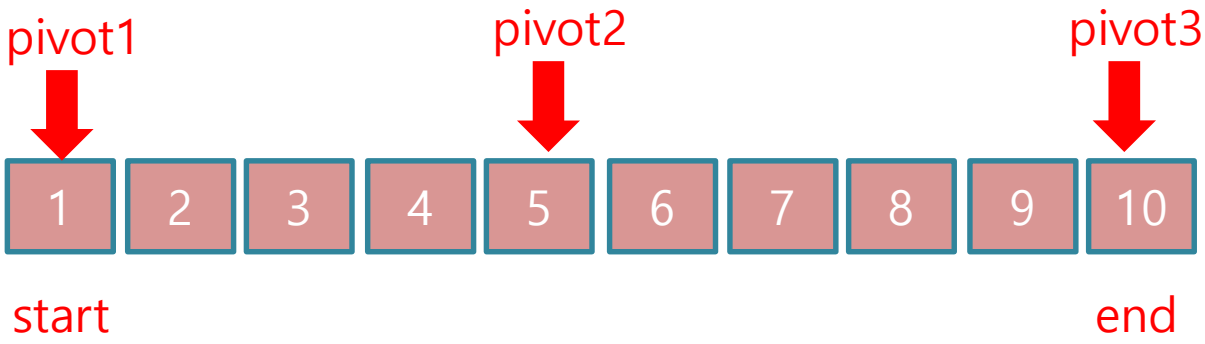
```
def get_pivot(data, start, end):
```



Pivot 인덱스에서의 값을 정렬하기 위해 버블 정렬을 사용

정렬 알고리즘
: Quicksort

```
def get_pivot(data, start, end):
```

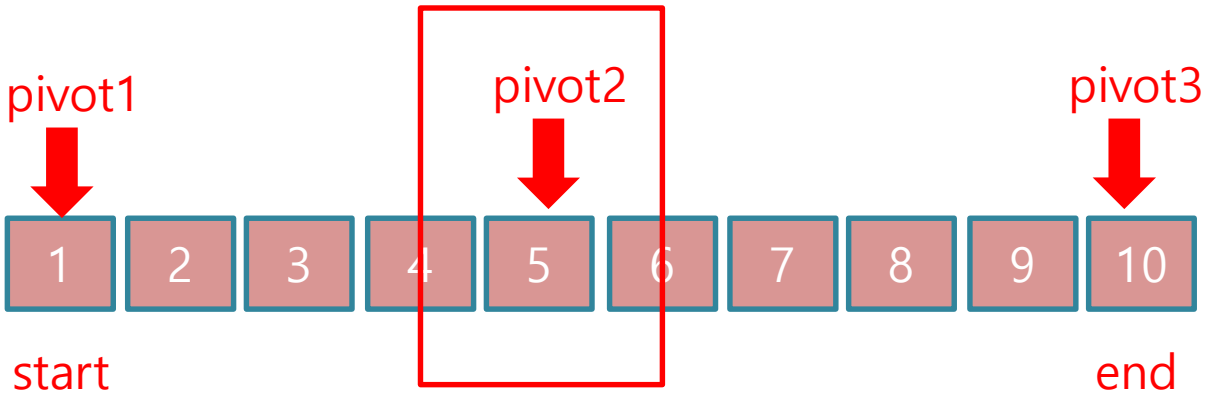



```
def get_pivot(data, start, end):
    mid = (start + end)//2
    list_idx = [start, mid, end]

    #bubble sort
    if data[list_idx[0]] > data[list_idx[1]]:
        list_idx[0], list_idx[1] = list_idx[1], list_idx[0]
    if data[list_idx[1]] > data[list_idx[2]]:
        list_idx[1], list_idx[2] = list_idx[2], list_idx[1]
    if data[list_idx[0]] > data[list_idx[1]]:
        list_idx[0], list_idx[1] = list_idx[1], list_idx[0]

    return list_idx[1]
```

버블정렬

가운데 값인 5의 인덱스 mid를 반환할 것

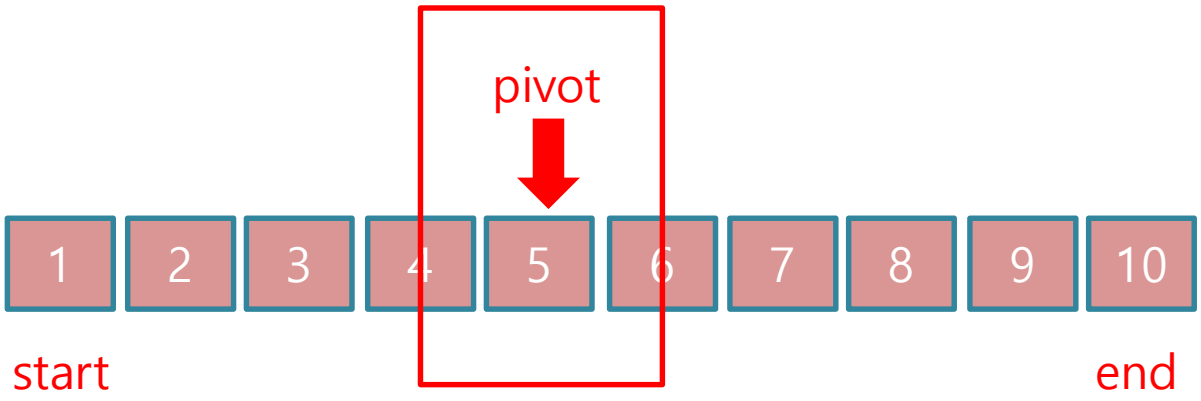
정렬 알고리즘
: Quicksort

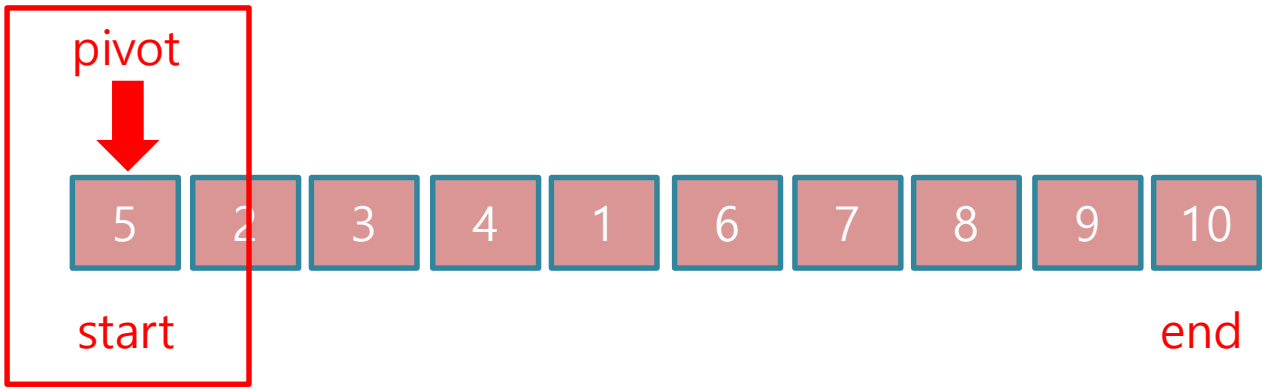


그럼 중간 값 pivot을 구했으니 코드를 다시 짜야 하나요?

정렬 알고리즘
: Quicksort

Pivot을 맨 왼쪽으로 옮기면 됩니다.





정렬 알고리즘
: Quicksort

```
def partition(data, start, end):
    global call_of_partition
    call_of_partition += 1
    get_pivot 함수로 가운데 값을 얻고
    idx_pivot = get_pivot(data, start, end)
    data[start], data[idx_pivot] = data[idx_pivot], ₩
    맨 앞쪽으로 옮긴다(바꾼다)                    data[start]

    pivot = data[start]

    low = start + 1
    high = end
```

Quicksort는 정렬 알고리즘 중 성능이 꽤 좋다고 알려져 있고
Worst case로 빅오를 따지는 다른 경우와 다르게 average case로 계산합니다