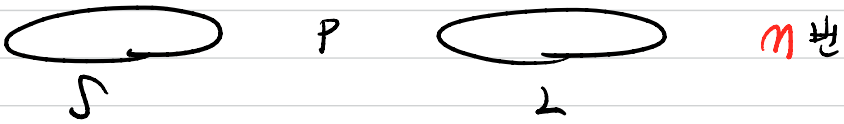
퀵 정렬

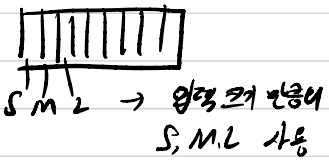


* Pseudo Code

```
def quick_sort(A):
    if |A| <= 1 : return A
    p = A[0]
    S, M, L = [], [], []
    for x in A:  → tail = A[1:] # 피벗을 제외한 리스트
        if p > x :
            S.append(A)
        elif p < x :
            L.append(x)
        else :
            M.append(x)        # 여러 개일 때
    return quick_sort(S) + M + quick_sort(L)
```

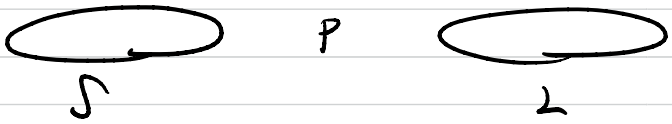
n-1회 비교.

① inplace가 아니다 ② stable O



* 종료조건 : 현재 리스트의 데이터 갯수가 1개인 경우

- 수행시간



비교횟수 : $T(n) \Rightarrow T(|S|) + T(|L|) + cn$

Worst case : 한쪽으로 몰린 경우 (pivot 보다 한 크기만 작은 경우)

$|S|$ or $|L| = 0$

$T(n) = T(n-1) + cn$

$= O(n^2)$

Best case : $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn$

$= 2T\left(\frac{n}{2}\right)$

$= O(n \log_2 n)$

In-place Quick Sort Algorithm

```
def quick_sort(array, start, end):
    if start >= end:
        return
    pivot = start
    left = start+1
    right = end
    while left <= right:
        # 피벗보다 큰 데이터를 찾을 때까지 반복
        while left <= end and array[left] <= array[pivot]:
            left += 1
        # 피벗보다 작은 데이터를 찾을 때까지 반복
        while right > start and array[right] >= array[pivot]:
            right -= 1

        if left > right:  # 엇갈렸다면 작은 데이터와 피벗을 교체
            array[right], array[pivot] = array[pivot], array[right]
        else:  # 엇갈리지 않았다면 작은 데이터와 큰데이터 교체
            array[left], array[right] = array[right], array[left]

    # 분할 후 왼쪽 부분과 오른 쪽 부분에서 각각 정렬을 수행.
    quick_sort(array, start, right-1)
    quick_sort(array, right+1, end)

quick_sort(arrg, 0, len(array)-1)
```

〈과정〉

5 7 9 0 3 1 6 2 4 8

- I 파트

Pivot = 5,

① (5) 7 9 0 3 1 6 2 4 8
Pivot

② (5) 4 9 0 3 1 6 2 7 8
Pivot

③ (5) 7 2 0 3 1 6 9 4 8
Pivot

⇄ 두 값이 엇갈린 경우: 작은 데이터와 Pivot의 위치 변경

- 분할 완료 -

1 7 2 0 3 [5] 6 9 4 8
Pivot

II 파트

[1] 7 2 0 3
Pivot

III 파트

⇄

[6] 9 7 8
Pivot