퀵 정렬 (Quick Sort)

> **분할 정복** 방법을 통해 정렬
>
> pivot을 정한 뒤 피봇의 위치를 확정해가며 정렬

특징

- 어떠한 기준으로 피 봇의 위치를 결정하느냐에 따라 큰 속도 차이가 발생
  - 항상 맨 왼쪽, 오른쪽, 중앙은 피봇을 설정하는 연산을 줄여 속도를 단축 했지만, 최악의 경우 O(n^2)
  - 랜덤 피봇 선정, 평균적인 속도를 보여줌.
    - 좋은 피봇인지 안좋을지 운에 맡김
  - 피봇의 변형에 따라 퀵소트의 여러 형태가 있다.
    - 멀티 피봇

진행 순서

1. 범위를 잡고
2. 배열 가운데에서 하나의 원소를 고름. ⇒ Pivot
3. Pivot기준 앞에는 값이 작은 원소, 뒤에는 값이 큰 원소로 나눈다. ⇒ 분할
4. 움직임이 끝나면 피봇이 중간으로 위치 스왑
5. 재귀로 위 과정을 반복

코드

```
void quickSort(int arr[], int left, int right) {
    if(left < right) {
        int pivot = partition(arr, left, right); // 정복

        // 분할
```

```
        quickSort(arr, left, pivot-1);
        quickSort(arr, pivot+1, right);
    }
}

void partition(int arr[], int left, int right) {
    int pivot = arr[right];
    int i = left - 1;

    for (int j = left; j <=right-1; j++) {
        if(arr[j] <= pivot){
            i++;
            swap(&arr[i], &arr[j]);
        }
    }

    swap(&arr[i + 1], &arr[right]);
    return (i + 1);
}
```

시간 복잡도

> 원소가 정렬되있든, 아니든 모두 탐색함

최선

$$O(NlogN)$$

평균

$$O(NlogN)$$

최악의 경우는 배열이 오름차순이나 내림차순 정렬이 되어있을 때, 순환 호출의 깊이가 깊어짐

$$O(N^2)$$

공간 복잡도

> 한개의 배열 안에서 수행

$$O(N)$$

장점

- 불필요한 데이터의 이동을 줄이고 먼거리의 데이터를 교환할 뿐만 아니라, 한번 결정된 피벗들은 추후 연산에서 제외되는 특성 때문에, 시간 복잡도가 O(nlogn)이다.
- 정렬하고자 하는 배열 안에서 교환하는 방식이므로 다른 메모리 공간이 별도로 필요하지 않다.

단점

- 불안정 정렬(Unstable Sort)
- 정렬된 배열에 대해서 Quick Sort의 불균형 분할에 의해 오히려 수행시간이 더 많이 걸린다.
