O(n log n), 기타 정렬 & 안정/불안정 정렬

O(n log n)의 시간 복잡도를 가지는 정렬

퀵 정렬

- 분할정복 (한 개의 큰 문제를 두 개의 작은 문제로 분할하는 식으로 접근)
  - **특정한 값(피벗)을 기준으로 큰 숫자와 작은 숫자를 서로 교환한 뒤에 배열을 반으로 나눈다!**
- **피벗(Pivot)**
  - 정렬 시 사용하는 **기준 값** (보통 처음이나 끝 값, 중위값을 사용)
- 평균적으로 O(nlogn), 최악의 경우에는 O(n^2)의 시간 복잡도
  - 영역을 분할해 정렬을 하기 때문에 더욱 빠르다!
- **불안정(unstable)** 정렬
  - 중복 데이터가 입력 순서와 동일하게 유지되지 않음!

[3, 7, 8, 1, 5, 9, 6, 10, 2, 4] 퀵 정렬로 정렬하기

1. [3, 7, 8, 1, 5, 9, 6, 10, 2, 4] → 3이 기준점
2. [3, **2**, 8, 1, 5, 9, 6, 10, **7**, 4] → 왼쪽으로부터 큰 값과 오른쪽으로부터 작은 값을 찾아 서로 바꿔준다. (2 ↔ 7)
3. [3, 2, **1**, **8**, 5, 9, 6, 10, 7, 4] → 다시 왼쪽으로부터 큰 값과 오른쪽으로부터 작은 값을 찾아 서로 바꿔준다. (1 ↔ 8)
4. [1, 2, **3**, 8, 5, 9, 6, 10, 7, 4] → 오른쪽에서 3보다 작은 값이 없으므로 3과 1의 위치를 서로 바꿔준다.(엇갈림)
   a. 3의 위치가 정렬되고 ,3을 기준으로 왼쪽에 있는 값은 3보다 작고 오른쪽에 있는 값은 3보다 커진다 → **분할**
   b. 3의 위치가 정렬됐으므로 다시 분할된 영역에서 각각 피벗값을 정해주고 정렬을 진행한다. → [1, 2, **3**, 8, 5, 9, 6, 10, 7, 4]
5. [**1**, 2, **3**, 8, 5, **4**, 6, 10, 7, **9**]

a. 1보다 작은 값이 없으므로 1의 위치 확정

b. 피벗(8)을 기준으로 큰 값과 작은 값인 4와 9의 위치를 바꿔준다. 9 ↔4

6. [**1**, **2**, **3**, 8, 5, 4, 6, **7**, **10**, 9]

   a. 2는 영역에 자기밖에 없으므로 2의 위치 고정

   b. 피벗(8)을 기준으로 큰 값과 작은 값인 7과 10의 위치를 바꿔준다. 10 ↔ 7

…..

7. [**1, 2, 3, 4, 6, 7, 8, 9, 10**] → **정렬 완료**

O(n^2)인 퀵정렬(최악의 케이스)

- 이미 정렬되어있는 배열, 피벗이 최댓값이나 최솟값일 때
  - [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    → 1을 피벗으로 잡고 진행하면 모든 수마다 피벗을 잡고 탐색을 하게 되기 때문에 시간 복잡도가 O(n^2)가 된다!

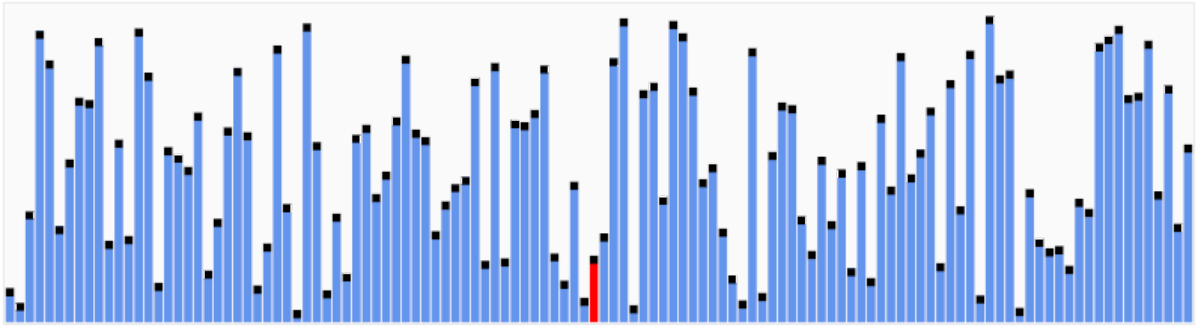
퀵 정렬 구현하기

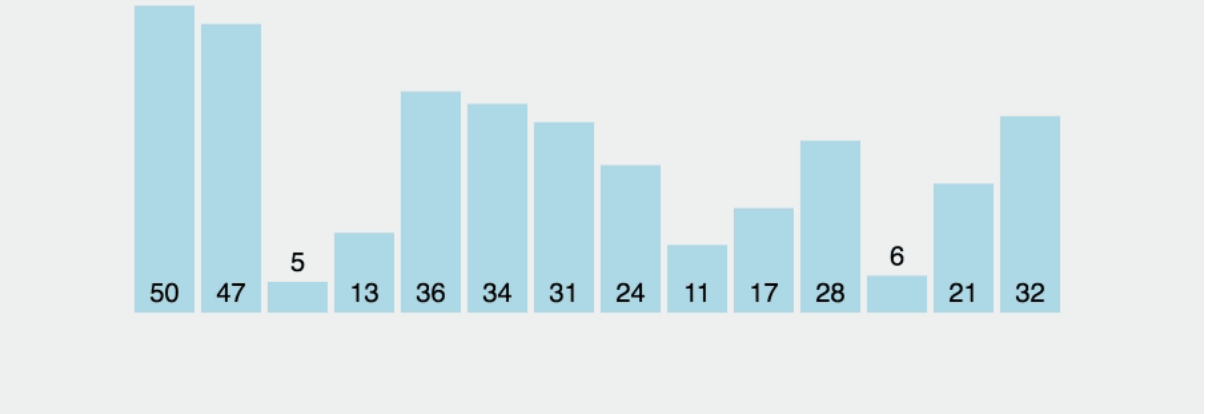
```
def quick_sort (arr, start, end):
    # 원소가 1개인 경우 함수 종료
    if start >= end:
        return
    # 첫 번째 원소를 피벗으로 설정
    pv = start
    # 좌측 리스트 시작점
    left = start + 1
    # 우측 리스트 시작점
    right = end

    while left <= right:

      # 피벗보다 큰 값을 찾을 때까지 무한 반복문
        while left <= end and arr[left] <= arr[pv]:
            left += 1

      # 피벗보다 작은 값을 찾을 때까지 문한 반복문
        while right > start and arr[right] >= arr[pv]:
            right -= 1

      # 탐색하는 데이터 위치가 다른 경우(엇갈린 경우)
        if left > right:
            arr[right], arr[pv] = arr[pv], arr[right]
        else:
            arr[left], arr[right] = arr[right], arr[left]
    # 분할 이후 좌측 및 우측 리스트 각각에에 대해 퀵 정렬 수행
    quick_sort(arr, 0, right -1)
```

```
    quick_sort(arr, right + 1, end)

quick_sort(arr, 0, len(arr)-1)
print(arr)
```

```
# Pythonic!
def quick_sort (arr):
    # 리스트 내 원소가 1개인 경우 함수 종료
    if len(arr) <= 1:
        return arr

    # 첫 번째 원소를 피벗으로 설정
    pv = arr[0]
    # 피벗을 제외한 리스트
    tail = arr[1:]

    # 분할된 좌측 리스트
    left_list = [x for x in tail if x <= pv]
    # 분할된 우측 리스트
    right_list = [x for x in tail if x > pv]

    # 분할 이후 좌측 및 우측 리스트 각각에에 대해 퀵 정렬 수행
    return quick_sort(left_list) + [pv] + quick_sort(right_list)

print(quick_sort(arr))
```

그림으로 보는 퀵 정렬



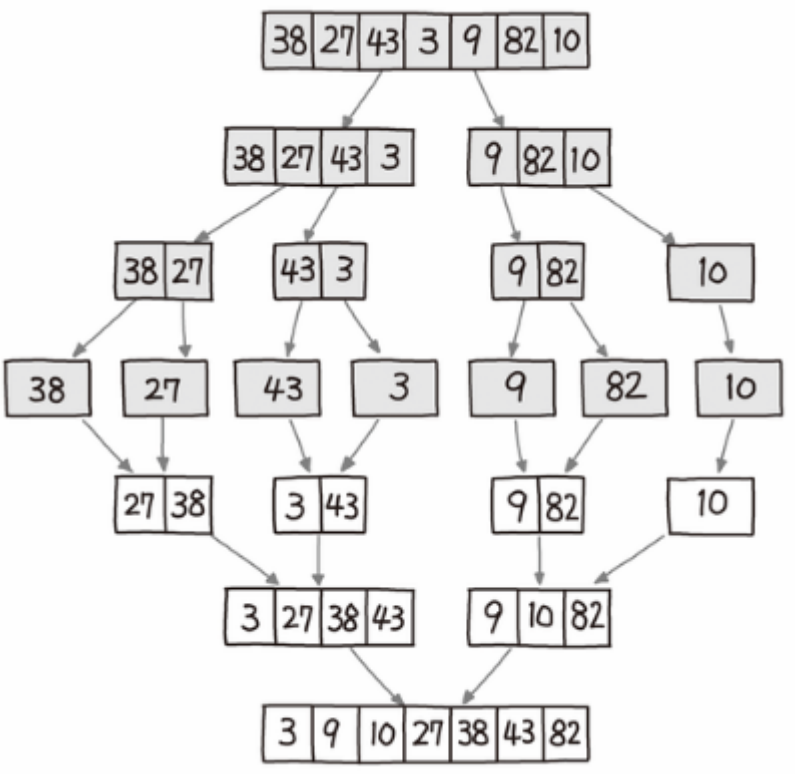


노란색 - pivot, 초록색 - pivot보다 작은 값, 보라색, pivot보다 큰 값, 주황색 - 정렬완료 된 값

병합정렬(merge sort)

- 분할 정복
  - 원소 개수가 0이나 1이 될 때까지 **값을 쪼갠 뒤 병합하면서 정렬**하는 방법
- **안정(stable)** 정렬
  - 중복 데이터가 입력 순서와 동일하게 유지!
- 평균적인 성능은 퀵정렬에 비해 조금 밀림, 병합된 정렬을 저장하기 위한 임시 배열 필요 ⇒ **공간복잡도 증가**!

그림으로 보는 병합 정렬



병합정렬 구현하기

```
def merge_sort(arr):
    # 분할된 배열의 길이가 2보다 작아질 때까지
    if len(arr) < 2:
```

```
        return arr

    # 배열을 분할한다.
    mid = len(arr) // 2
    low_arr = merge_sort(arr[:mid])
    high_arr = merge_sort(arr[mid:])

    # 정렬한(병합한) 배열들을 담을 임시 배열
    merged_arr = []
    # 분할된 배열들을 정렬하기 위한 인덱스
    l = h = 0
    while l < len(low_arr) and h < len(high_arr):
        #작은 값을 배열에 먼저 넣어준다.
        if low_arr[l] < high_arr[h]:
            merged_arr.append(low_arr[l])
            l += 1
        else:
            merged_arr.append(high_arr[h])
            h += 1
    merged_arr += low_arr[l:]
    merged_arr += high_arr[h:]
    return merged_arr
```

그 외 정렬 방법

팀 정렬(Tim sort)

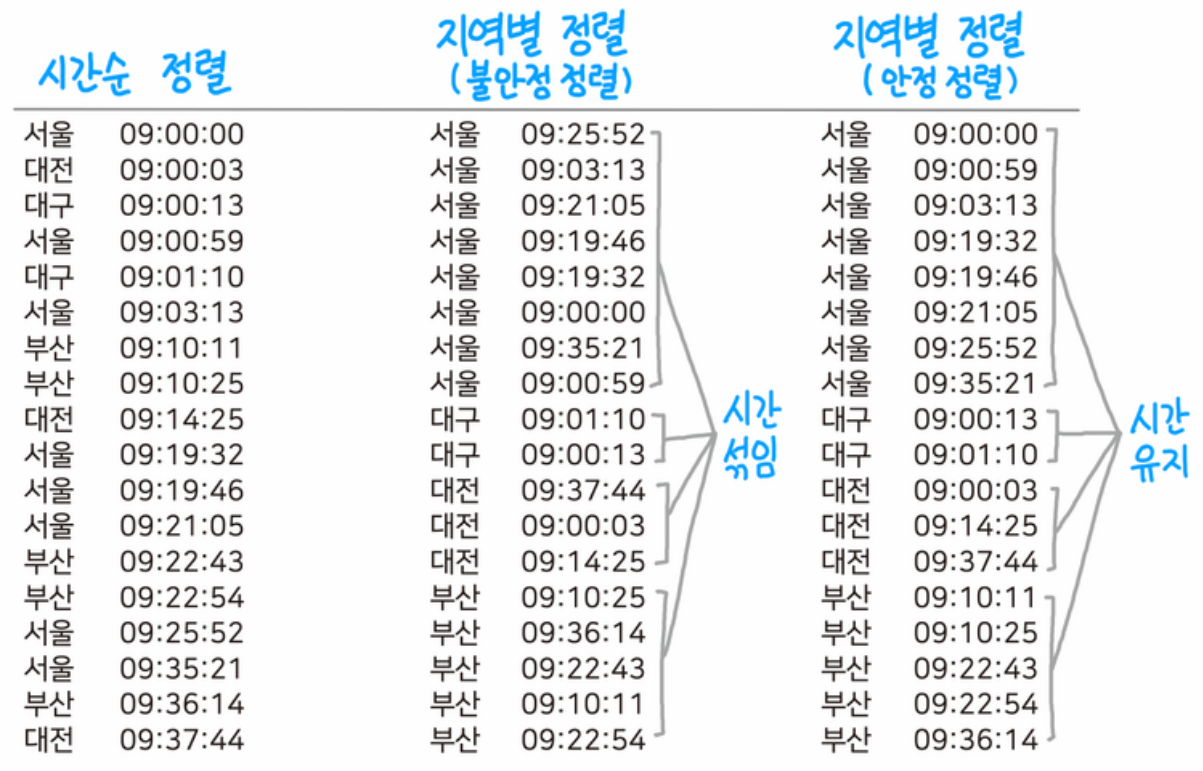
- 선택 정렬 + 병합 정렬
- **Python에서 활용**하는 정렬 방식!
- 일반적으로 O(n log n)의 시간복잡도를 가진다

팀 정렬의 접근 방식

1. 현실세계의 데이터들은 완전 무작위로 배열돼 있기 보단 어느 정도 정렬된 상태로 배열돼 있는 경우가 많지 않을까?
2. 그렇다면 정렬을 해야하는 전체 배열을 작은 덩어리들로 잘라 각각의 덩어리를 Insertion sort로 정렬한 뒤 merge sort로 병합하면 좀 더 빠르지 않을까?

안정 정렬과 불안정 정렬

- 중복된 값이 **입력된 순서와 동일하게 유지**되는가?!



정렬 문제 모음!!

실제 코테에서는 정렬 알고리즘을 직접 구현할 일이 적음!!(싸피는….ㅠㅠ)

직접 정렬 알고리즘을 구현하기보다는 배운 개념들을 어떻게 문제에 활용해서 시간 복잡도를 줄일 수 있을 지 고민하면서 풀어보자!!

https://www.acmicpc.net/problem/3151 합이 0 (골5)

https://www.acmicpc.net/problem/18114 블랙프라이데이 (골5)

https://www.acmicpc.net/problem/1755 숫자 놀이 (실4)

https://www.acmicpc.net/problem/1431 시리얼 번호 (실3)

https://www.acmicpc.net/problem/2012 등수 매기기 (실3)

https://www.acmicpc.net/problem/18870 좌표압축(실2)

https://www.acmicpc.net/problem/2108 통계학(실3)

https://www.acmicpc.net/problem/11497 통나무 건너뛰기(실1)

https://www.acmicpc.net/problem/14729 칠무해 (실5)