chapter4. 퀵 정렬

- Hello Coding 그림으로 개념을 이해하는 알고리즘 -

TABLE OF CONTENTS

1. 퀵 정렬

병합 정렬과 마찬가지로 퀵 정렬도 분할 정복 기법과 재귀를 사용한 알고리즘입니다.

쉬운 이해를 위해서 다음과 같이 1부터 7까지 총 7개의 숫자가 들어있는 배열을 기준으로 설명하겠습니다.

```
[6, 5, 1, 4, 7, 2, 3]
```

항상 정 가운데를 기준으로 분할을 하는 병합 정렬과 달리, 퀵 정렬은 흔히 **피봇(pivot)**이라고 불리는 임의의 기준값을 사용합니다.
pivot 값을 선택하는데는 여러가지 방법이 있지만 여기서는 간단한 설명을 위해 정 중앙에 위치한 4을 pivot으로 정하겠습니다.
그리고 다음과 같이 이 pivot 값을 기준으로 pivot보다 작은 값의 그룹과 pivot보다 큰 값의 그룹으로 나눕니다.

위와 같이 pivot 값보다 작은 값들은 모두 왼편으로 몰고, 큰 값들은 모두 오른편으로 몰면 기준값은 정확히 정렬된 위치에 놓이게 됩니다.

또한 이런 방식으로 분할을 해놓으면 앞으로 더 이상 왼편에 있는 값들과 오른편에 있는 값들 간에는 비교를 할 필요가 없습니다.

따라서 반대편은 전혀 신경쓰지 않고 왼편이든 오른편이든 같은편 내의 값들 끼리만 비교 후 정렬을 할 수 있게 됩니다.

먼저 왼편을 동일한 방식으로 정렬해보도록 하겠습니다.

왼편의 정 가운데에 위치한 pivot 값인 2 보다 작은 값인 1인 왼쪽에 큰 값인 3은 오른쪽에 위치시켰습니다.

이제 양쪽 모두 값이 하나씩 밖에 없기 때문에 이로써 왼편의 정렬 작업은 완료되었습니다

```
      p
[1] < 2 < [3]
```

오른편도 동일한 방식으로 정렬해보겠습니다. 오른편의 pivot 값인 5 보다 작은 값은 없으므로 7과 6을 모두 오른편에 위치시켰습니다.

```
      p
[] < 5 < [7, 6]
```

오른편의 오른편(?)에는 값이 2개가 있기 때문에 추가 정렬이 필요합니다. 왼편에는 값이 없지만 오른편에는 여전히 두 개의 값이 있기 때문에, 동일한 방식의 정렬을 적용하겠습니다.

```
      p
[6] < 7 < []
```

마지막으로 지금까지 좌우로 분할했던 값들을 모두 합쳐보면 다음과 같이 정렬된 배열을 얻을 수 있습니다.

```
[1, 2, 3, 4, 5, 6, 7]
```

지금까지 살펴본 것과 같이 퀵 정렬은 배열을 pivot 값 기준으로 더 작은 값과 큰 값으로 반복적으로 분할하여 정렬해나가는 방식을 취하고 있습니다.

2. 빅오표기법 복습

퀵 정렬은 피봇을 어떤 원소를 선택했느냐에 따라 처리속도가 달라집니다.

이진 탐색 : O(Log n) 제일빠름
단순 탐색: O(n)
퀵 정렬 : O(n Log n)
선택 정렬 : O(n의 제곱) 느림

병합 정렬이라는 정렬 알고리즘도 있는데 실행 시간은 O(n Log n) 입니다.

퀵 정렬이랑 병합 정렬이랑 실행 시간이 똑같죠?

그런데 퀵 정렬은 최악의 경우 O(n의제곱)이 걸릴 수도 있지만 병합정렬은 항상 O(n Log n)입니다.

그럼 퀵 정렬을 뭐하러 쓰나요?

빅오 표기법에서 상수항은 문제가 되는 경우가 드물어서 보통 무시합니다.

하지만 가끔은 이 상수 때문에 차이가 발생하기도 합니다.

퀵 정렬과 병합 정렬이 그 예입니다.

퀵 정렬이 병합 정렬보다 더 작은 상수를 가집니다. 그래서 실행시간이 O(n Log n)으로 동일하다면 퀵 정렬이 더 빠릅니다.

그리고 퀵 정렬을 사용할 때 최악의 경우보다는 일반적인 경우가 훨씬 많이 발생하기 때문에 현실에서는 퀵 정렬이 더 빠릅니다.

결론 : 퀵 정렬 > 병합 정렬 (퀵 정렬 상수가 작아서)

3. 퀵 정렬 예제코드

```
def quicksort(array):
  if len(array) < 2:
    return array
  else:
    pivot = array[0]
    less = [i for i in array[1:] if i <= pivot]
    greater = [i for i in array[1:] if i > pivot]
    return quicksort(less) + [pivot] + quicksort(greater)

print(quicksort([10,5,2,3]))
```

```
[2, 3, 5, 10]
```

if len(array) < 2:

기본 단계 : 원소 개수가 0이나 1이면 이미 정렬되어 있는 상태

else:

재귀 단계

less, greater 변수에는 리스트 내포를 집어 넣음

리스트 내포 = [표현식 for i in 반복자 if~]

4. 핵심 내용 요약

★분할 정복은 문제를 더 작은 조각으로 나누어 푸는 것을 말합니다. 만약 리스트에 분할 정복을 적용한다면 기본 단계는 원소가 없는 빈 배열이거나 하나의 원소만 가진 배열이 됩니다.

★퀵 정렬을 구현하려면 기준 원소를 무작위로 선택합니다. 퀵 정렬의 평균적인 실행 시간은 O(n Log n)입니다.

★빅오 표기법에서 가끔씩 상수가 중요해질 때도 있습니다. 퀵정렬이 병합 정렬보다 빠른 이유도 상수 때문입니다.

★단순 탐색과 이진 탐색을 비교할 때는 상수항이 전혀 문제가 되지 않습니다. 왜냐하면 리스트가 길어지면 O(n Log n)이 O(n)보다 훨씬 빨라지기 때문입니다.