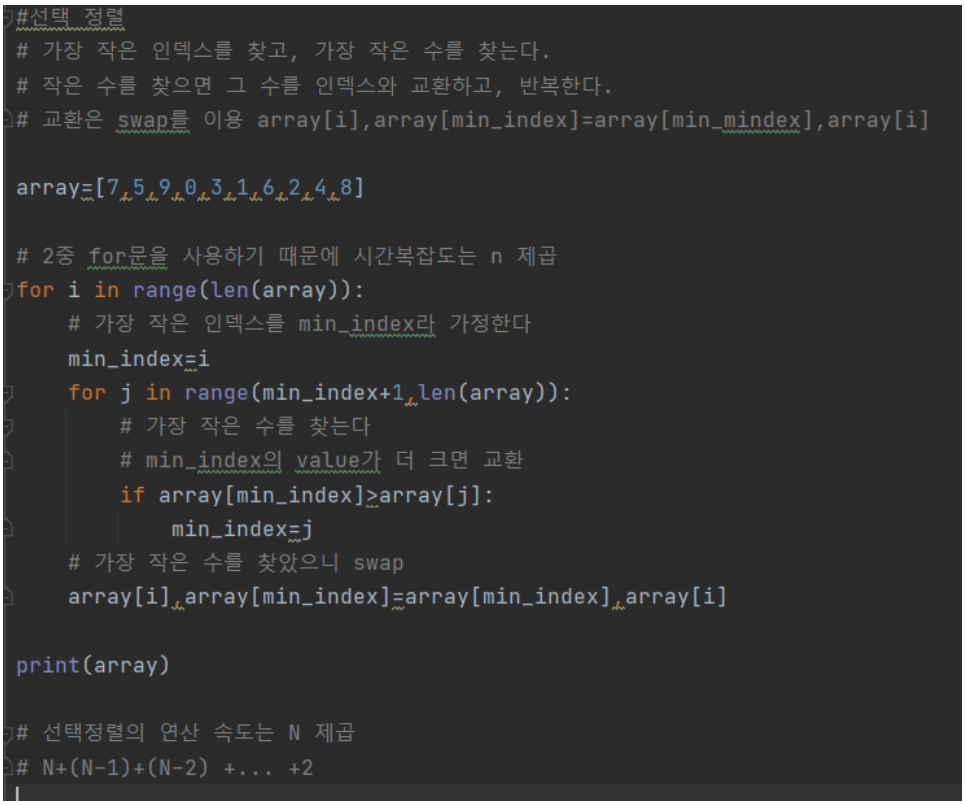
정렬

- 정의 - 데이터를 특정한 기준에 따라 순서대로 나열 (오름차순, 내림차순)
- 종류 - 선택 정렬, 삽입 정렬, 퀵 정렬, 계수 정렬 ...

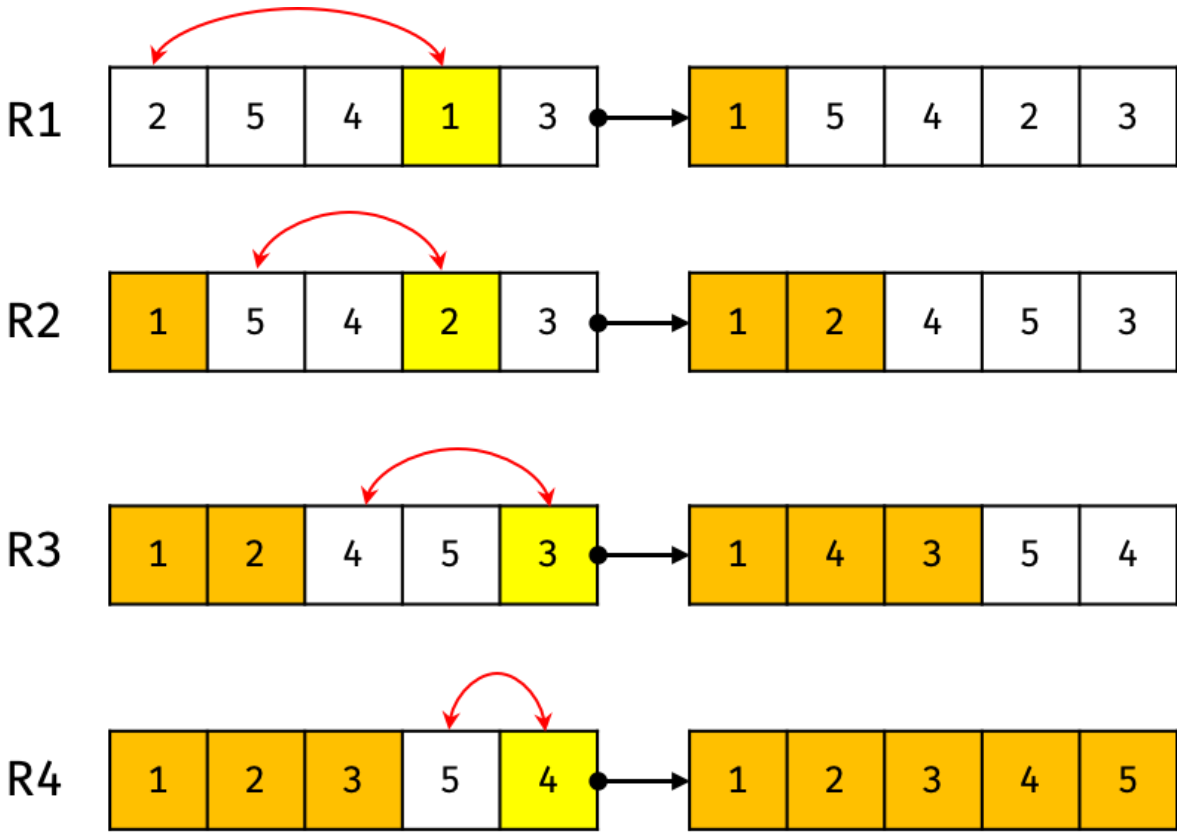
선택 정렬

- 가장 작은 데이터를 **선택하여 맨 앞에 있는 데이터와 바꾸고,** 그 다음 작은 데이터를 선택해 앞에서 두 번째 데이터와 바꾸는 과정 등을 반복하여 정렬
- 순서
  1. 가장 작은 값과 인덱스를 찾는다.
  2. 맨 앞에 있는 것과 교환한다. (위치 변경)
  3. 그 다음으로 작은 인덱스를 찾고 1번째와 교환한다.
  4. N-1번 반복한다



```python
#선택 정렬
# 가장 작은 인덱스를 찾고, 가장 작은 수를 찾는다.
# 작은 수를 찾으면 그 수를 인덱스와 교환하고, 반복한다.
# 교환은 swap를 이용 array[i],array[min_index]=array[min_mindex],array[i]

array=[7,5,9,0,3,1,6,2,4,8]

# 2중 for문을 사용하기 때문에 시간복잡도는 n 제곱
for i in range(len(array)):
    # 가장 작은 인덱스를 min_index라 가정한다
    min_index=i
    for j in range(min_index+1,len(array)):
        # 가장 작은 수를 찾는다
        # min_index의 value가 더 크면 교환
        if array[min_index]>array[j]:
            min_index=j
    # 가장 작은 수를 찾았으니 swap
    array[i],array[min_index]=array[min_index],array[i]

print(array)

# 선택정렬의 연산 속도는 N 제곱
# N+(N-1)+(N-2) +... +2
```

선택 정렬 python code



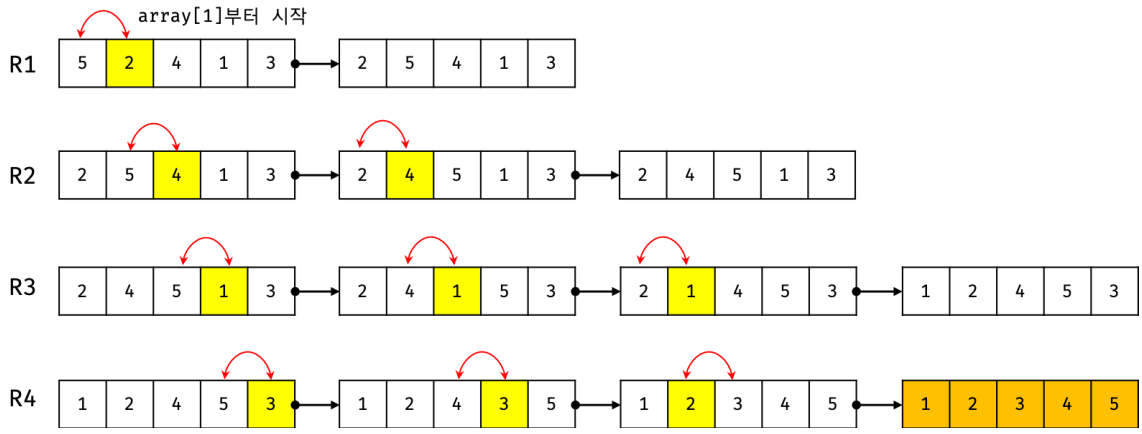
선택 정렬 순서, 시간 복잡도는 n^2

삽입 정렬

- 특정한 데이터를 적절한 위치에 **삽입한다는 의미에서 삽입 정렬이라고 부른다.**
- 특정한 데이터가 적절한 위치에 들어가기 이전에, 그 앞 까지의 데이터는 이미 정렬되어 있다고 가정한다.
- 장점은 필요할 때만 위치를 바꾸므로 데이터가 거의 정렬되어 있을 때 선택정렬보다 훨씬 효율적이다.
- 순서
    1. 두 번째 데이터부터 시작한다. (첫 번째 데이터는 그 자체로 정렬되어 있다고 판단)
    2. 두 번째 데이터를 첫 번째 데이터와 비교하고 두 번째 데이터가 작다면 첫 번째 데이터의 왼쪽, 크다면 오른쪽으로 위치한다. (여기서 중요한 점은 숫자와 숫자 사이에 공간에 넣는다는 점에서 삽입 정렬이라고 한다)
    3. 나머지 데이터 (N-2)도 반복한다.

```python
# 삽입정렬
# 1번 인덱스(2번째)부터 시작해서 삽입하는 식으로 정렬
# 2중 for 문 사용
# 만약 거의 정렬되어 있는 상태로 주어진다면 break 문으로 빠져나오기 때문에 O(N)의 시간 복잡도를 가질수 있음

array=[7,5,9,0,3,1,6,2,4,8]

for i in range(1,len(array)):
    for j in range(i,0,-1):
        if array[j]<array[j-1]:
            array[j],array[j-1]=array[j-1],array[j]
        else:
            break
print(array)
```

삽입 정렬 python code



삽입 정렬 순서도, 시간 복잡도는 n^2

퀵 정렬

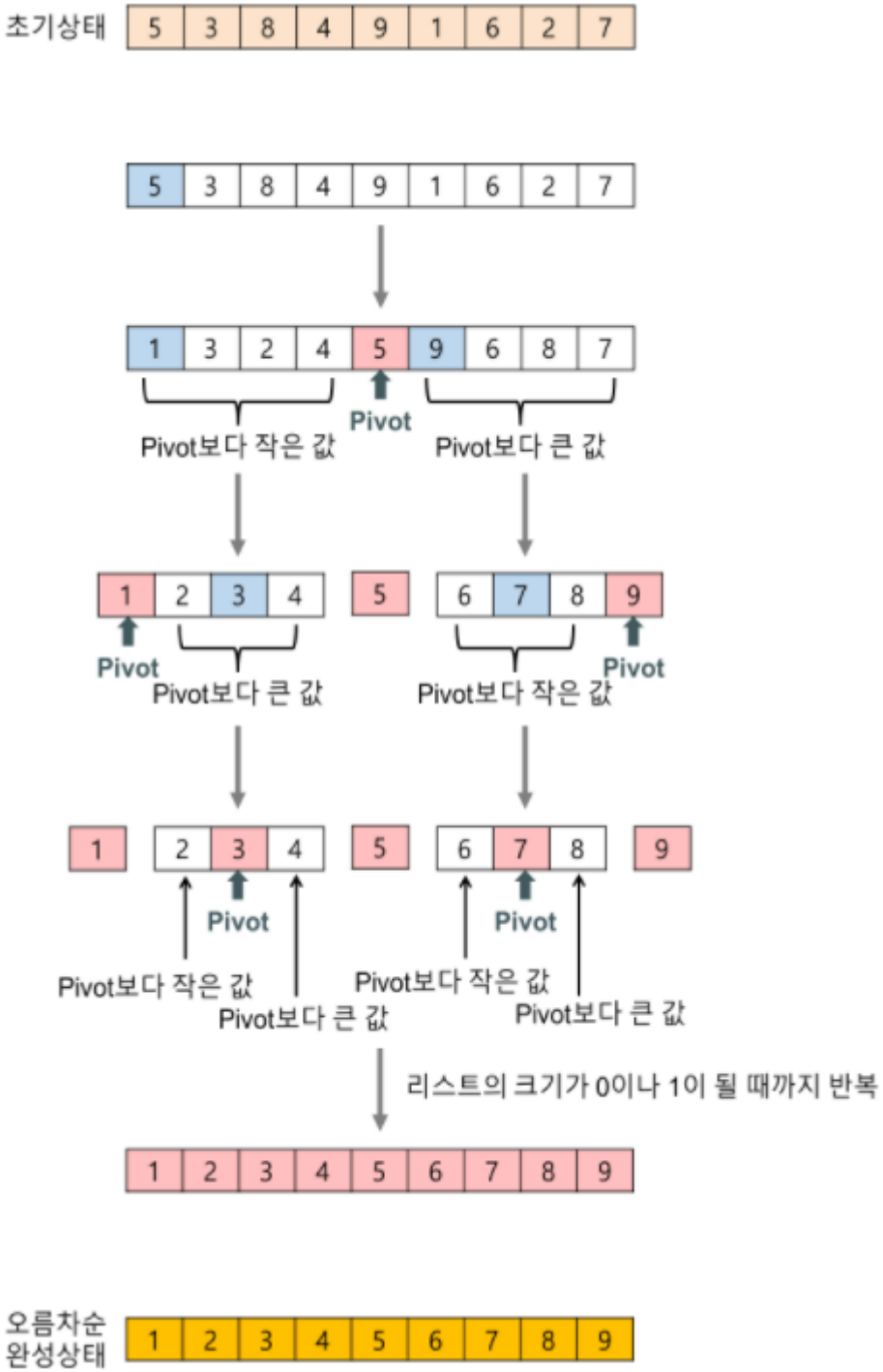
- 기준을 설정한 다음 큰 수와 작은 수를 교환한 후 리스트를 반으로 나누는 방식으로 동작
- **기준이 되는 것을 Pivot**이라고 한다.
- 여기서는 호어 분할 (Hoare Partion)으로 피벗을 설정
- 순서
    1. 리스트의 첫 번째 데이터를 피벗으로 설정하고, 왼쪽에서는 피벗보다 큰 데이터를 선택, 오른쪽에서는 피벗보다 작은 데이터를 선택한다.
    2. 선택된 두 데이터의 위치를 변경한다.

3. 그 다음 피벗보다 큰 데이터와 작은 데이터를 각각 찾고 두 데이터의 위치를 변경한다.
4. 만약 서로 엇갈린다면 작은 데이터와 피벗의 위치를 변경한다.
5. 피벗이 변경되었다면 분할이 완료된 것으로 피벗을 기준으로 왼쪽과 오른쪽 리스트로 분할한다.
6. 마찬가지로 왼쪽 리스트와 오른쪽 리스트도 같은 방식으로 진행한다.

- **퀵 정렬은 재귀 함수와 동작 원리가 같다**. 재귀 함수로 구현 되어 있을 때 간결하게 구현 가능
- 퀵 정렬이 끝나는 조건은 리스트의 데이터 개수가 1개인 경우

```python
array = [5,7,9,0,3,1,6,2,4,8]

def quick_sort(array, start, end):
    if start >= end: # 원소가 1개인 경우 종료
        return
    pivot = start # 피벗은 첫 번째 원소
    left = start + 1
    right = end
    while(left <= right):
        # 피벗보다 큰 데이터를 찾을 때까지 반복
        # 현 상황에선 피벗보다 큰 데이터가 없기 때문에 +2
        while(left <= end and array[left] <= array[pivot]):
            print('none')
            left += 1
        # 피벗보다 작은 데이터를 찾을 때까지 반복

        while(right > start and array[right] >= array[pivot]):
            print('right')
            right -= 1
        if(left > right): # 엇갈렸다면 작은 데이터와 피벗을 교체
            print('here')
            array[right], array[pivot] = array[pivot], array[right]
        else: # 엇갈리지 않았다면 작은 데이터와 큰 데이터를 교체
            array[left], array[right] = array[right], array[left]
    # 분할 이후 왼쪽 부분과 오른쪽 부분에서 각각 정렬 수행
    quick_sort(array, start, right - 1)
    quick_sort(array, right + 1, end)

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

퀵 정렬 python code

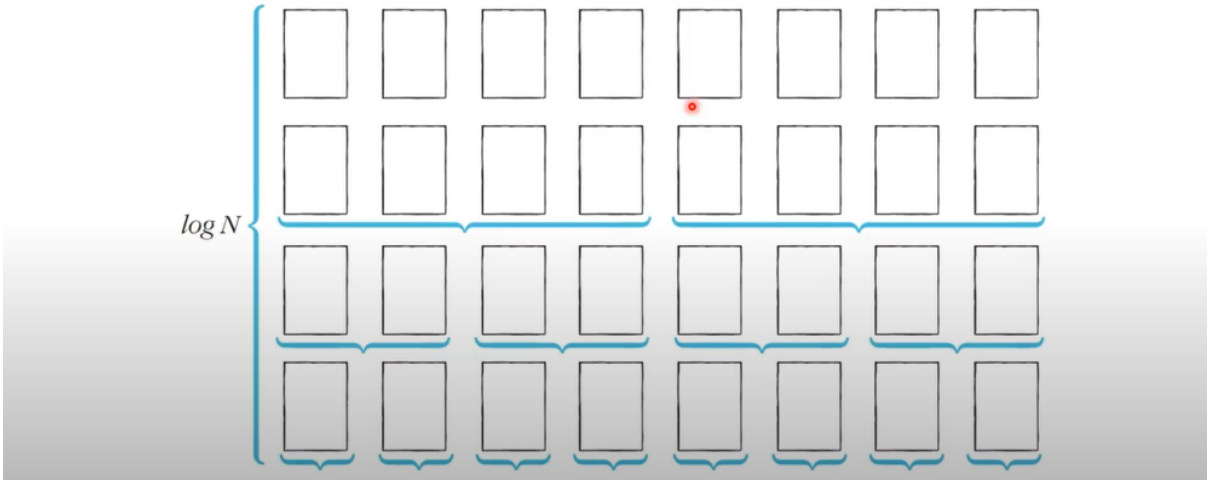


퀵 정렬 순서도, 시간 복잡도는 최악의 경우 n^2, 평균 시간 복잡도는 $NlogN$

- 퀵 정렬의 시간 복잡도가 $NlogN$ 인 이유는 다음과 같다.

이상적인 경우 분할이 절반씩 일어난다면 전체 연산 횟수로 $O(NlogN)$를 기대할 수 있습니다.

- **너비 X 높이** = $N \times logN = NlogN$



- 최악의 경우 시간 복잡도가 N^2인 이유
  - 가장 왼쪽 데이터를 피벗으로 삼을 때, 이미 데이터가 정렬되어 있는 경우에는 매우 느리게 동작하기 때문이다.
  - 삽입 정렬은 이미 데이터가 정렬되어 있는 경우에는 매우 빠르게 동작

계수 정렬

- 특정한 조건이 부합할 때만 사용할 수 있는 매우 빠른 정렬 알고리즘
- 모든 데이터가 양의 정수인 상황
- 데이터의 개수는 N, 데이터 중 최대값이 K일때, 최악의 경우에도 수행 시간 O(N+K)를 보장
- 중복되는 데이터가 많을 때, 가장 큰 값과 가장 작은 값의 차이가 100만이 넘지 않을 때 효과적으로 사용이 가능
- 데이터의 리스트를 활용하여 정렬

```python
# 계수 정렬
# 특정한 조건이 부합할 때만 사용할 수 있는 매우 빠른 정렬 알고리즘
# 반복되는 데이터 개수가 N개 일때 이를 오름차순이나 내림차순으로 정렬할때 사용
# 가장 큰 데이터와 가장 작은 데이터 차이가 100만이 안넘어 갈때 효과적으로 사용
# 계수 정렬은 모든 범위를 담을 수 있는 크기의 리스트(배열)을 선언
# 가장 큰 데이터와 가장 데이터의 범위가 모두 담길 수 있는 하나의 리스트 생성
# 예를들면 가장 작은 것은 0, 큰 것은 9라면 모든 범위가 포함될수 있는 리스트인 10이 필요 ==> 이를 0으로 다 초기화
# 데이터를 하나씩 확인하여 인덱스의 데이터를 1씩 증가

array=[7,0,3,1,6,2,1,4,8,0,2]
# 최대 값 +1 만큼 list 만들어준다
count=[0]*(max(array)+1)

for i in range(len(array)):
    # 여기가 핵심, count 안에 변수로 array[i]로 넣고 이에 대한 인덱스를 증가
    count[array[i]]+=1

# 정보 확인
for i in range(len(count)):
    for j in range(count[i]): #반복되는 과정
        print(i,end=" ") # 여기에서 출력은 i로 한다

print(count)
```

계수 정렬 Python Code

- **정렬할 데이터**: 7 5 9 0 3 1 6 2 9 1 4 8 0 5 2

인덱스	0	1	2	3	4	5	6	7	8	9
개수(Count)	2	2	2	1	1	2	1	1	1	2

계수 정렬 순서

파이썬 정렬 라이브러리

- sorted() 함수를 제공, 최악의 경우에도 시간 복잡도 O(NlogN)을 보장
-

```python
# sort 함수
# return 값이 없고 원본값 수정
array=[7,0,3,1,6,2,1,4,8,0,2]
array.sort()
print("sort1:",array)
a2=array.sort() #return 값이 없음, 원본 값만 수정 가능
print("sort1:",a2)

# sorted 함수
# 원본값은 그대로 이고, 정렬된 값을 반환 return 값이 존재
array=[7,0,3,1,6,2,1,4,8,0,2]
result=sorted(array)
print("sort2",result)
print("sort2",array)

# sorted(), sort()는 key를 매개변수로 입력받아 정렬이 가능, k
array=[('바나나',2),('사과',5),('당근',3)]
def setting(data):
    return data[1]
result=sorted(array,key=setting)
kk=array.sort(key=setting)
print(result)
print(array)
```

시간 복잡도

이름	Best(최상)	Avg(평균)	Worst(최악)
삽입 정렬 (Insertion Sort)	n	n^2	n^2
선택 정렬 (Selection Sort)	n^2	n^2	n^2
버블 정렬 (Bubble Sort)	n^2	n^2	n^2
퀵 정렬 (Quick Sort)	$n\log_2 n$	$n\log_2 n$	n^2
힙 정렬 (Heap Sort)	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$
병합 정렬 (Merge Sort)	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$
계수 정렬 (Counting Sort)	n + k	n + k	n + k

문제 풀이