4. 정렬 알고리즘

강의상 순서는 4번이지만 수업 진도가 정렬부터 배웠기 때문에 정렬 부터 시작

정렬이란? 데이터를 특정한 기준에 따라 순서대로 나열하는 것

일반적으로 문제 상황에 따라서 적절한 정렬 알고리즘이 공식처럼 사용된다.

1. 선택정렬

처리되지 않은 데이터 중에서 **가장 작은 데이터를 선택해 맨 앞에 있는 데이터와 바꾸**는 것을 반복

전체 연산 순서는

N + N-1 + N-2 + ... 2

등차수열 합과 빅오표기법에 따라 시간복잡도는 O(N^2)

▼ 코드

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

for i in range(len(array)):
    min_index = i #가장 작은 원소의 인덱스
    for j in range(i+1,len(array)):
        if array[min_index] > array[j]:
            min_index = j
    array[i], array[min_index] = array[min_index], array[i]
    # 파이썬에서는 위와 같은 스왑 연산을 한 줄로 간단히 작성가능
print(array)
```

2. 삽입정렬

처리되지 않은 데이터를 하나씩 골라 적절한 위치에 삽입.

선택 정렬에 비해 구현 난이도가 높은 편이지만 일반적으로 더 효율적!

▼ 코드

```
array = [5,1,8,3,0,4,6,2,7,9]
# 맨처음 원소. 인덱스 0의 원소는 정렬되어있다고 생각하고 출발해서 range가 1부터
for i in range(1,len(array)):
```

```
    for j in range(i,0,-1): #i부터 1까지 1씩 감소하며반복
        if array[j] < array[j-1]: # 한 칸씩 왼쪽으로 이동
            array[j], array[j-1] = array[j-1], array[j]
        else: # 자기보다 작은 데이터를 만나면 해당 위치에서 멈춤
            break
print(array)
```

선택정렬과 마찬가지로 반복문이 두번 중첩되어 사용됨. 시간 복잡도 O(N^2). 중첩 반복문 안에서 기본적인 연산인 스왑만 이루어짐.

삽입 정렬은 현재 데이터가 거의 정렬되어 있는 상태일 때 매우 빠르게 동작. 최선의 경우 O(N)의 시간복잡도. 왼쪽 데이터와 비교했을 떄 바로 멈추기 때문에 단순 상수시간만큼 걸리게된다.

3. 퀵정렬

이름에서 볼 수 있는 것 처럼 빠른 정렬 알고리즘 중 하나. 일반적인 상황에서 가장 많이 사용되는 정렬 방법.

기준 데이터를 설정하고 그 기준보다 큰 데이터와 작은 데이터의 위치를 바꾸는 정렬방법.

병합정렬과 더불어 python, java 등 대부분의 언어의 정렬 라이브러리의 근간이 되는 알고리즘.

가장 기본적인 퀵 정렬은 첫 번째 데이터를 기준 데이터인 Pivot으로 설정한다.

- 퀵정렬 과정

설정한 피봇을 기준으로 왼쪽 끝 요소의 인덱스를 pl, 오른쪽 끝 요소의 인덱스를 pr. 각각 왼쪽커서, 오른쪽 커서 라고 함.

이제 배열을 두 그룹으로 나눌 것임

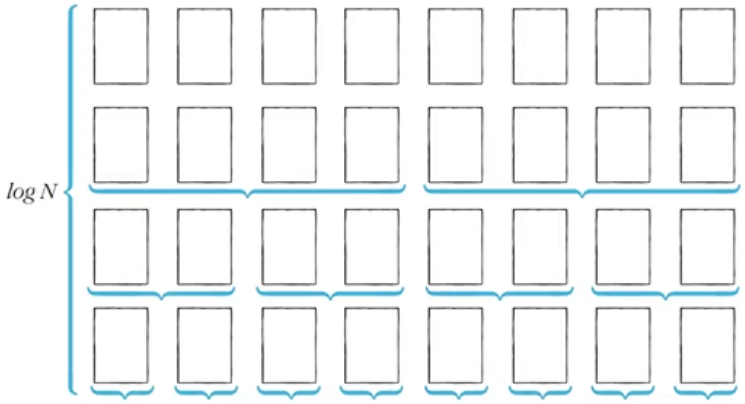
arr[pl] ≥ pivot, arr[pr]≤ pivot 을 찾을때까지 각각 오른쪽 왼쪽으로 스캔. 찾으면 arr[pl]과 arr[pr]의 값을 교환하고 계속 스캔 진행

두 커서가 교차하게 될때까지 진행. 교차하면 그룹을 나누는 과정이 끝나게 되고 나눠진 각각의 두 배열에서 다시 퀵정렬 진행!

이렇게 재귀적으로 진행됨. 배열을 분할(devide or partition)한 후 각각의 배열에서 또다시 퀵정렬.

퀵 정렬이 빠른 이유: 직관적인 이해

- 이상적인 경우 분할이 절반씩 일어난다면 전체 연산 횟수로 $O(NlogN)$를 기대할 수 있습니다.
  - **너비 X 높이** = $N \times logN = NlogN$



평균의 경우 O(NlogN), 최악의 경우O(N^2) - 첫번째 원소를 피봇으로 설정하게 되면 분할한 왼쪽 부분이 없는 상태로 오른쪽만 가지고 계속 진행하게 되기 때문.

일반적으로 퀵정렬을 가지고 구현한 프로그래밍 언어들의 정렬 라이브러리는 NlogN의 시간복잡도를 가지도록 구현됨.

▼ 기본 퀵정렬 코드(python 간단방법)

```
# 마찬가지로 첫번째 원소를 피봇으로 하는 퀵정렬
# 근데 파이썬의 문법으로 훨씬 간단하게 작성하는 방법
# 슬라이싱과 리스트 컴프리헨션 이용!

arr = [5,1,8,3,0,4,6,2,7,9]
def quick_sort(array):
    # 리스트가 하나 이하의 원소만을 담고 있다면 종료
    if len(array) <= 1:
        return array

    pivot = array[0] # 피벗은 첫 번째 원소
    tail = array[1:] # 피벗을 제외한 리스트

    left_side = [x for x in tail if x <= pivot] # 분할된 왼쪽 부분
    right_side = [x for x in tail if x > pivot] # 분할된 오른쪽 부분

    # 분할 이후 왼쪽 부분과 오른쪽 부분에서 각각 정렬을 수행하고, 전체 리스트를 반환
    return quick_sort(left_side) + [pivot] + quick_sort(right_side)

print(quick_sort(array))
```

4. 계수정렬(Counting Sort)

특정한 조건이 부합할 때만 사용할 수 있지만 매우 빠르게 동작하는 정렬 알고리즘.

⇒ 데이터의 크기 범위가 제한되어 정수 형태로 표현할 수 있을때 사용가능!

데이터의 개수가 N, 데이터(양수) 중 최댓값이 K일 때 최악의 경우에도 수행 시간 O(N+K)를 보장한다!

동작 예시

1. 가장 작은 데이터부터 가장 큰 데이터까지의 범위가 모두 담길 수 있는 리스트를 생성한다. (가장 큰 데이터 = 배열의 마지막 인덱스. 각 인덱스가 데이터의 값이 됨. → 각각의 데이터가 몇번씩 등장했는지를 구하기 위함)
2. 데이터를 하나씩 확인하며 데이터의 값과 동일한 인덱스의 데이터를 1씩 증가시킨다.
3. 결과 확인 - 리스트의 첫번째 데이터부터 하나씩 그 값만큼 반복하여 인덱스를 출력한다.

각각의 데이터가 몇번씩 등장했는지 count하는 방식으로 동작하는 정렬 알고리즘! 상대적으로 공간복잡도가 높지만 계수 정렬의 사용 조건만 만족한다면 수행 시간은 효율적!

시간복잡도와 공간복잡도 모두 O(N+K)

데이터가 만약 0과 999999 2개만 존재하는 경우 배열을 백만개 크기로 생성해야함. 매우 비효율적.

→ 동일한 값을 가지는 데이터가 여러 개 등장할 때 효과적!

▼ 코드

```
# 모든 원소의 값이 0보다 크거나 같다고 가정
array = [7, 5, 9, 0, 3, 1, 6, 2, 9, 1, 4, 8, 0, 5, 2]
# 모든 범위를 포함하는 리스트 선언 (모든 값은 0으로 초기화)
count = [0] * (max(array) + 1) #인덱스니까 배열의 최댓값 +1

for i in range(len(array)):
    count[array[i]] += 1 # 각 데이터에 해당하는 인덱스의 값 증가

for i in range(len(count)): # 리스트에 기록된 정렬 정보 확인
    for j in range(count[i]):
        print(i, end=' ') # 띄어쓰기를 구분으로 등장한 횟수만큼 인덱스 출력
```

- 정렬 알고리즘 비교

정렬 알고리즘 비교하기

- 앞서 다룬 네 가지 정렬 알고리즘을 비교하면 다음과 같습니다.
- 추가적으로 대부분의 프로그래밍 언어에서 지원하는 표준 정렬 라이브러리는 최악의 경우에도 $O(NlogN)$을 보장하도록 설계되어 있습니다.

정렬 알고리즘	평균 시간 복잡도	공간 복잡도	특징
선택 정렬	$O(N^2)$	$O(N)$	아이디어가 매우 간단합니다.
삽입 정렬	$O(N^2)$	$O(N)$	데이터가 거의 정렬되어 있을 때는 가장 빠릅니다.
퀵 정렬	$O(NlogN)$	$O(N)$	대부분의 경우에 가장 적합하며, 충분히 빠릅니다.
계수 정렬	$O(N+K)$	$O(N+K)$	데이터의 크기가 한정되어 있는 경우에만 사용이 가능하지만 매우 빠르게 동작합니다.

*퀵정렬 최악의 경우 시간복잡도 n^2인거 기억!

일반적으로 문제에서 정렬함수를 구현하도록 요구하지 않는다면 그냥 각 언어의 표준 정렬 라이브러리를 이용해서 정렬하는 것이 좋다!

▼ 예제1) 두 배열의 원소 교체

〈문제〉 두 배열의 원소 교체: 문제 설명

- 동빈이는 두 개의 배열 A와 B를 가지고 있습니다. 두 배열은 N개의 원소로 구성되어 있으며, 배열의 원소는 모두 자연수입니다.
- 동빈이는 **최대 K 번의 바꿔치기** 연산을 수행할 수 있는데, 바꿔치기 연산이란 배열 A에 있는 원소 하나와 배열 B에 있는 원소 하나를 골라서 두 원소를 서로 바꾸는 것을 말합니다.
- 동빈이의 최종 목표는 배열 A의 모든 원소의 합이 최대가 되도록 하는 것이며, 여러분은 동빈이를 도와야 합니다.
- N, K, 그리고 배열 A와 B의 정보가 주어졌을 때, 최대 K 번의 바꿔치기 연산을 수행하여 만들 수 있는 **배열 A의 모든 원소의 합의 최댓값을 출력**하는 프로그램을 작성하세요.

```
# <이것이 코딩테스트다> 정렬part
# <예제> - 두 배열의 원소 교체
```

```
#  내 풀이
N,K = map(int,input().split())
A = list(map(int,input().split()))
B = list(map(int,input().split()))

A.sort(); B.sort() # B.sort(reverse=True)해서 똑같이 인덱스 0으로 사용해도 될듯
for x in range(K):
    if A[0] < B[-1]:
        A[0], B[-1] = B[-1], A[0]
        A.sort()
        B.sort()
    else:
        break

print(sum(A))
```