

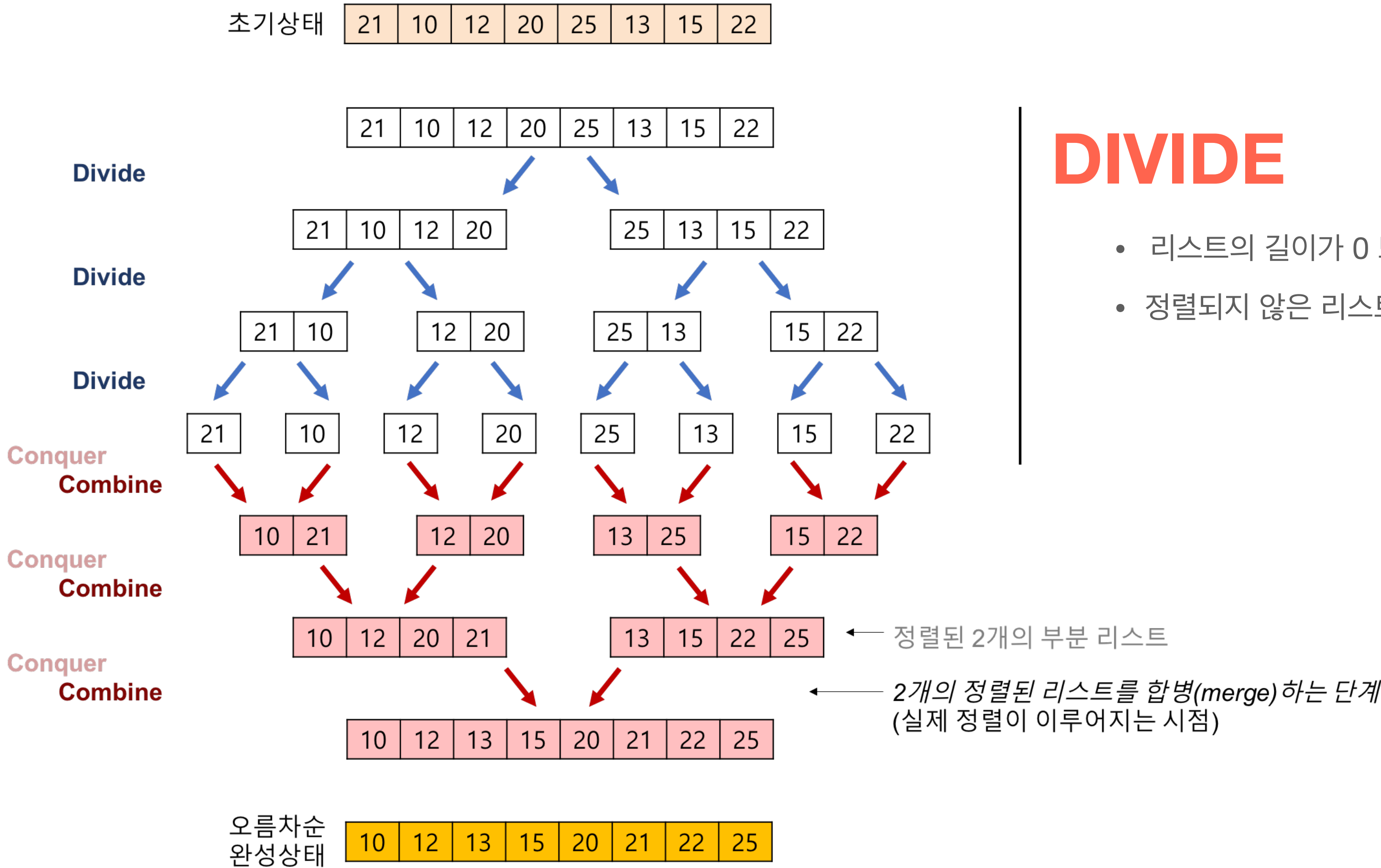
병합정렬

23.9.4 박송이

병합정렬 기초

- ‘존 폰 노이만(John von Neumann)’이라는 사람이 제안한 방법
- (분할 정복 알고리즘 중에 하나에 속한다)
- 분할 정복(divide and conquer) 방법
 - 문제를 작은 2개의 문제로 분리하고 각각을 해결한 다음, 결과를 모아서 원래의 문제를 해결하는 전략이다.
 - 분할 정복 방법은 대개 순환 호출을 이용하여 구현한다.
- 과정 설명
 - 리스트의 길이가 0 또는 1이면 이미 정렬된 것으로 본다. 그렇지 않은 경우에는
 - 정렬되지 않은 리스트를 절반으로 잘라 비슷한 크기의 두 부분 리스트로 나눈다.
 - 각 부분 리스트를 재귀적으로 합병 정렬을 이용해 정렬한다.
 - 두 부분 리스트를 다시 하나의 정렬된 리스트로 합병한다.

병합정렬 기초





DIVIDE

- 리스트의 길이가 0 또는 1이면 이미 정렬된 것으로 본다. 그렇지 않은 경우에는
- 정렬되지 않은 리스트를 절반으로 잘라 비슷한 크기의 두 부분 리스트로 나눈다.

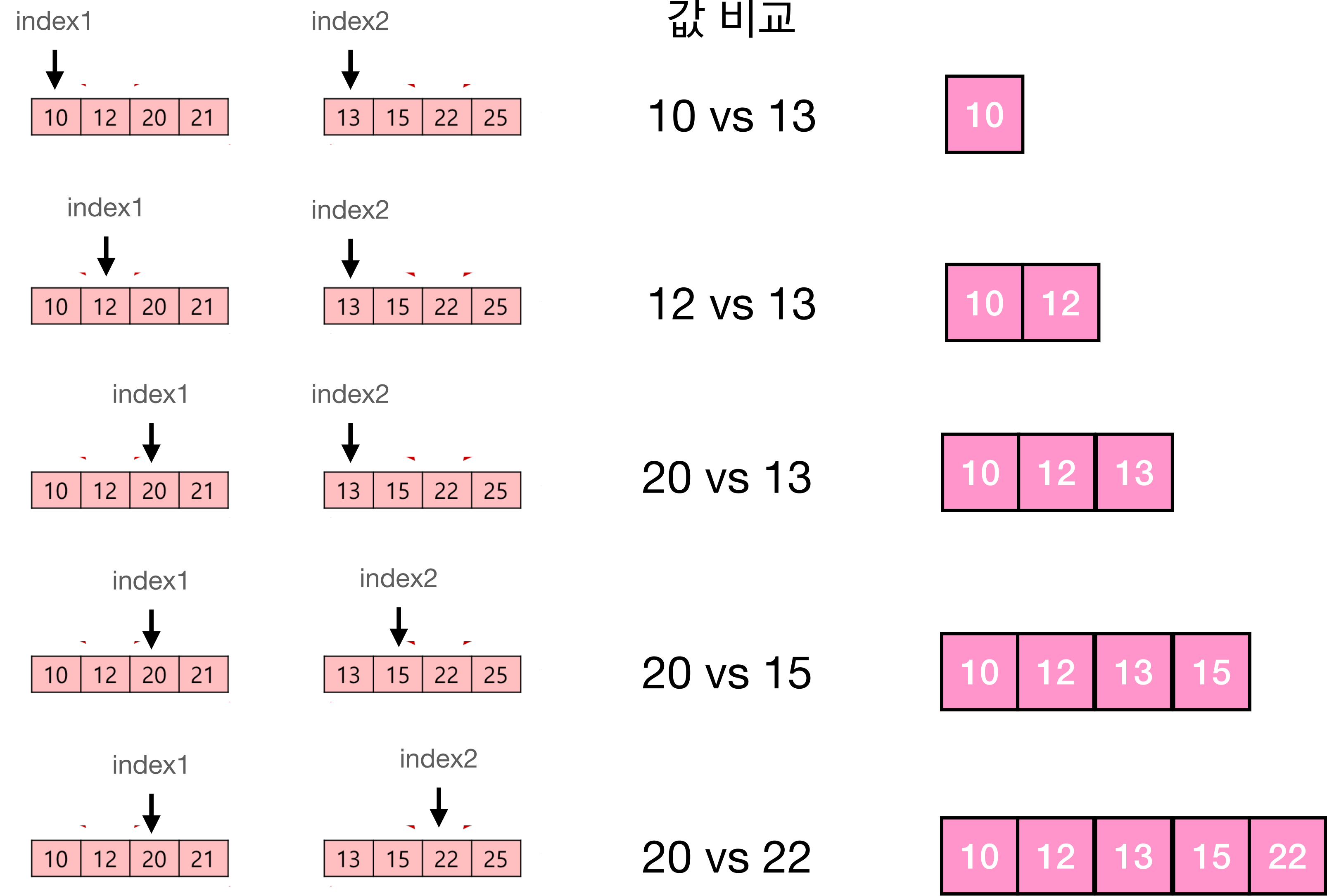
COMBINE

언어마다 다른 구현방식이 있지만
파이썬에서는 쪼개진 덩어리마다 포인터를 두고
포인터가 가리키는 값을 비교해 병합하면서 정렬한다.

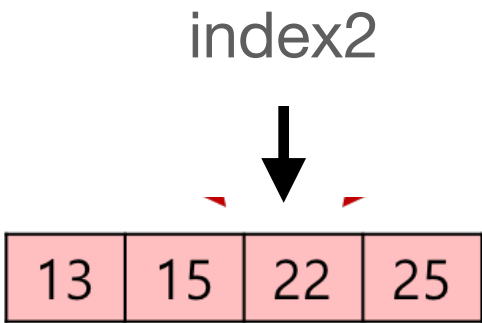
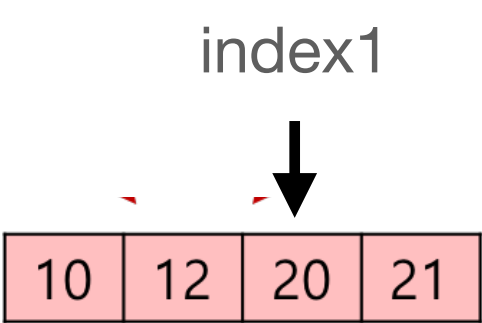
병합정렬 기초 - 전체 진행 과정

Iter	list name	list value	① merge_sort함수 수행 (분할)	② merge 함수 수행 (정렬 및 합병)	sorted list value
0	raw list	21, 10, 12, 20, 25, 13, 15, 22			
1	left list	21, 10, 12, 20			10, 12, 13, 15, 20, 21, 22, 25
1	right list	25, 13, 15, 22			
2	left list	21, 10			10, 12, 20, 21
2	right list	12, 20			
3	left list	21			10, 21
3	right list	10			
4	left list	12			12, 20
4	right list	20			
5	left list	25, 13			13, 15, 22, 25
5	right list	15, 22			
6	left list	25			13, 25
6	right list	13			
7	left list	15			15, 22
7	right list	22			

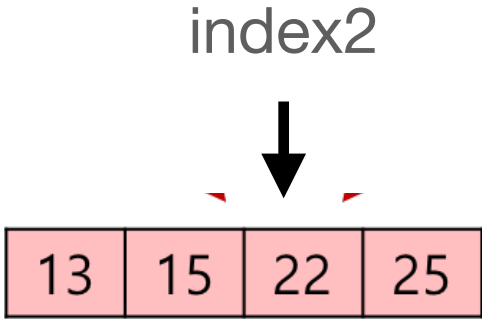
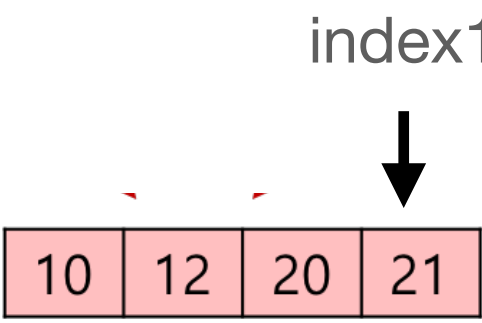
병합정렬 기초 - 병합 과정



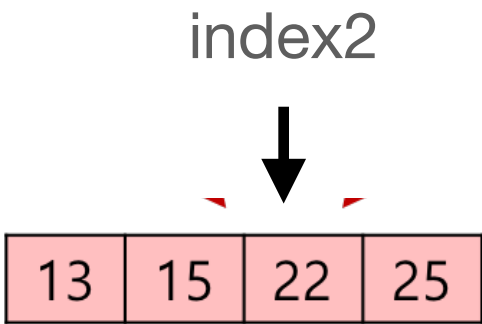
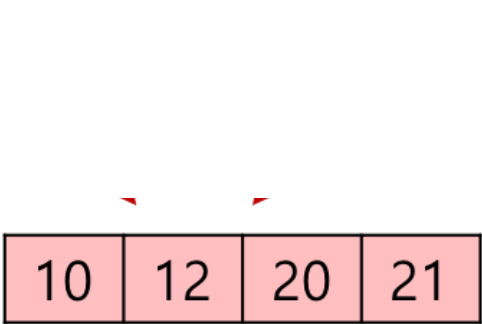
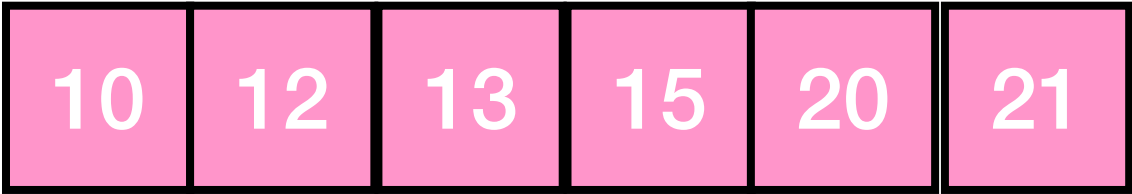
병합정렬 기초 - 병합 과정



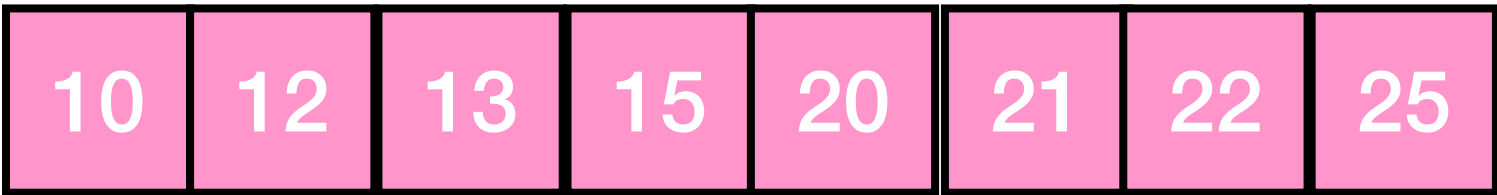
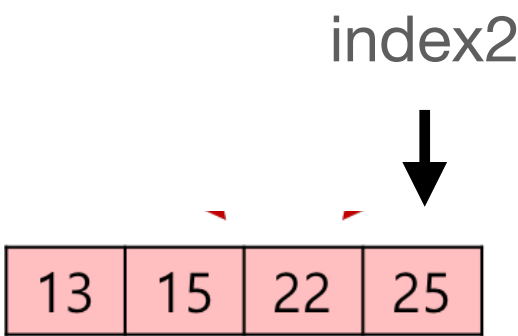
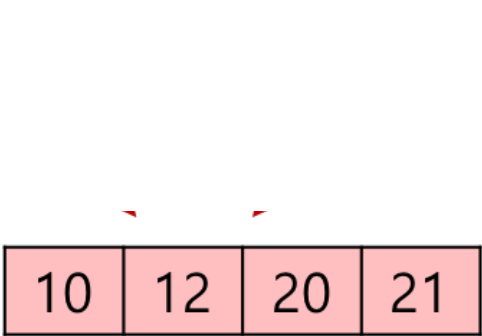
20 vs 22



21 vs 22



오른쪽 리스트는
이미 정렬이 되어
있으므로
값을 순서대로
가져옵니다.



병합정렬의 단점과 장점

- 단점
 - 만약 레코드를 배열(Array)로 구성하면, 임시 배열이 필요하다. (이 책에서는 tmp라는 이름의 배열을 임시로 사용한다)
 - 제자리 정렬(in-place sorting)이 아니다.
 - 레코드들의 크기가 큰 경우에는 이동 횟수가 많으므로 매우 큰 시간적 낭비를 초래한다.
- 장점
 - 안정적인 정렬 방법
 - 데이터의 분포에 영향을 덜 받는다. 즉, 입력 데이터가 무엇이든 간에 정렬되는 시간은 동일하다. ($O(n\log_2 n)$ 로 동일)
 - 만약 레코드를 연결 리스트(Linked List)로 구성하면, 링크 인덱스만 변경되므로 데이터의 이동은 무시할 수 있을 정도로 작아진다.
 - 따라서 크기가 큰 레코드를 정렬할 경우에 연결 리스트를 사용한다면, 합병 정렬은 퀵 정렬을 포함한 다른 어떤 정렬 방법보다 효율적이다.

다른 정렬 방법들과의 비교

정렬 알고리즘 시간복잡도 비교

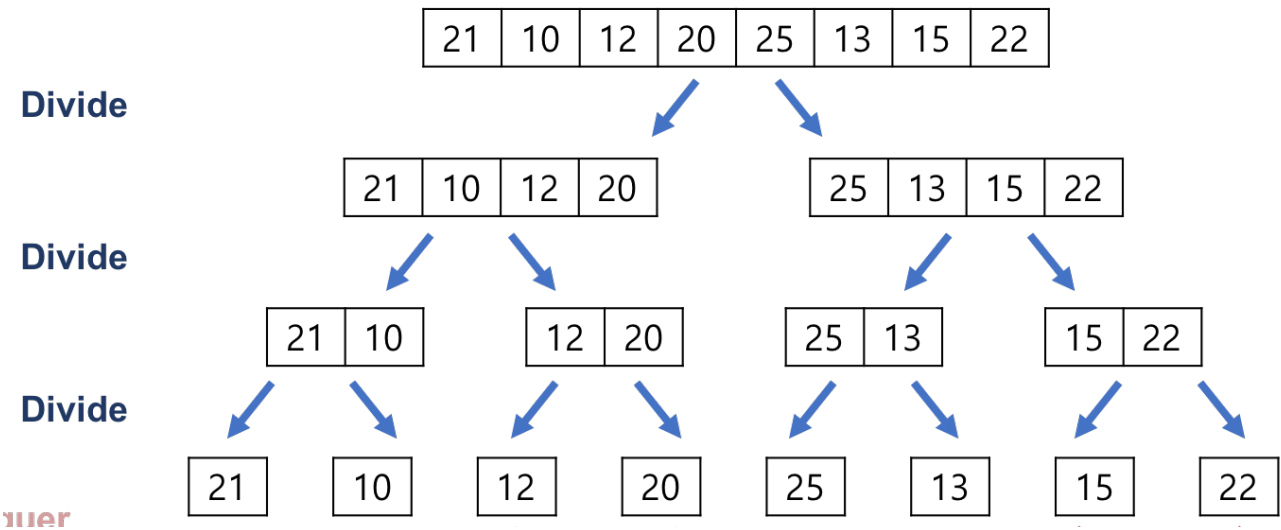
Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셸 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n \log_2 n$	$n \log_2 n$	n^2	0.014
힙 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.034
병합정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$	0.026

- 단순(구현 간단)하지만 비효율적인 방법
 - 삽입 정렬, 선택 정렬, 버블 정렬
- 복잡하지만 효율적인 방법
 - 퀵 정렬, 힙 정렬, 합병 정렬, 기수 정렬

위 포스팅과 관련된 자료 출처 : <https://gmlwjd9405.github.io/2018/05/06/algorithm-selection-sort.html>

문제 20번

기본적으로 입력이 1,000,000이므로 병합정렬로 해결해야 한다.



DIVIDE 부분 구현

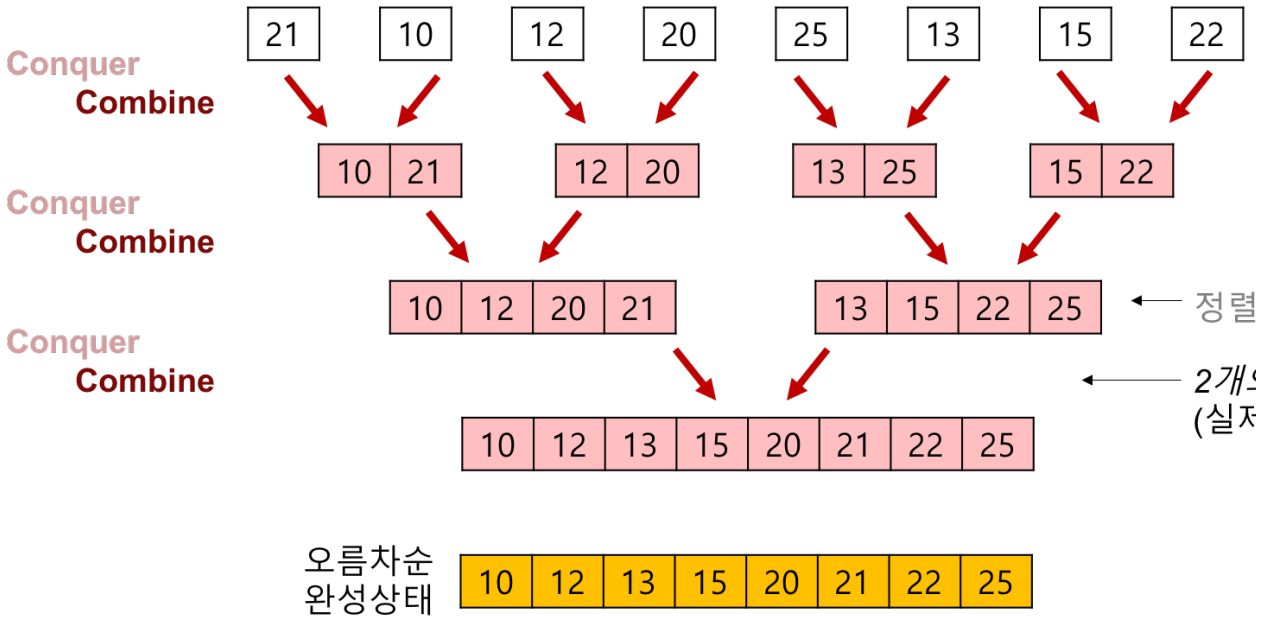
if e-s < 1 : return _____ 길이가 0 또는 1이라면 더 이상 분할하지 않는다.

m = int(s+(e-s) /2) _____ 길이가 2이상이라면 두개로 나누기 위해 중간 인덱스 값을 구해준다.

merge_sort(s,m) _____
merge_sort(m+1,e) _____ 처음 인덱스 ~ 중간 인덱스 / 중간 인덱스 +1 ~ 마지막 인덱스로 다시 나눠준다.

for i in range(s,e+1): _____
tmp[i] = A[i] _____ 원본 배열 A에 있던 값을 tmp 배열에 옮겨준다.

문제 20번



COMBINE 부분 구현

1)초기 세팅

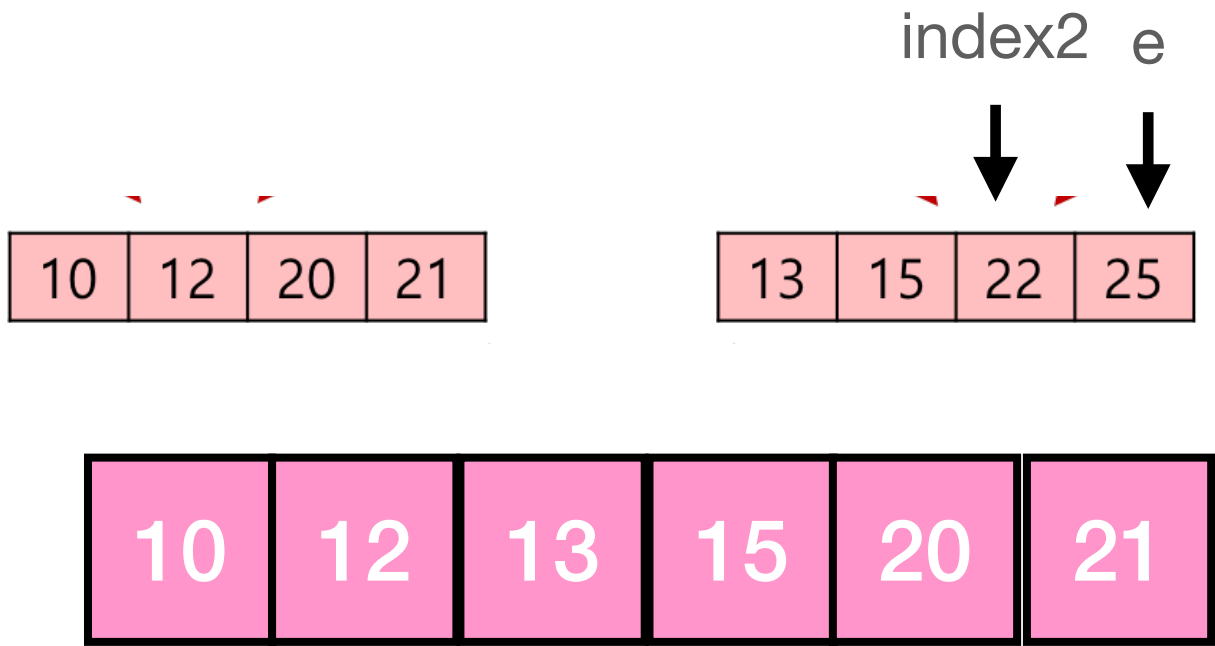
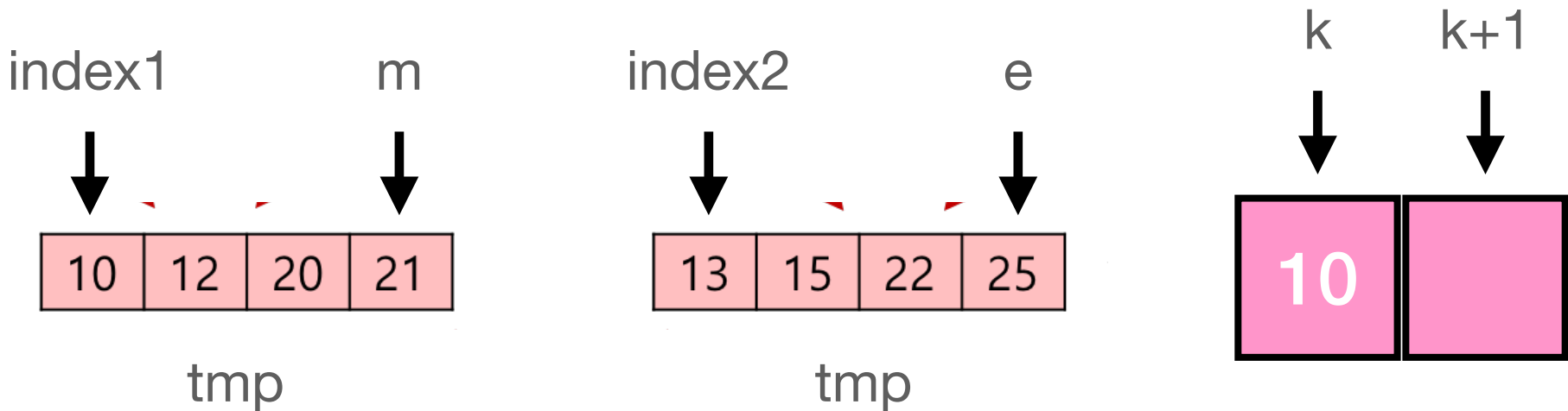
```
k = s
index1 = s
index2 = m+1
```

2) index가 가리키는 값 비교

```
while index1 <= m and index2 <= e:
    if tmp[index1] > tmp[index2]:
        A[k] = tmp[index2]
        k += 1
        index2 += 1
```

3) 정렬되지 않은 부분은 순차적으로 넣어주기

```
while index2 <= e:
    A[k] = tmp[index2]
    k += 1
    index2 += 1
```

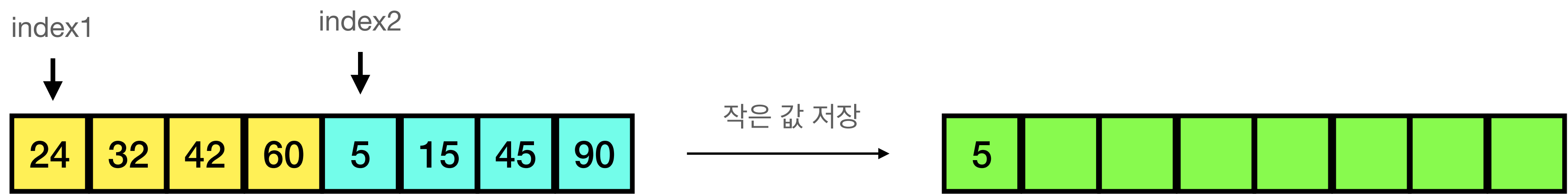
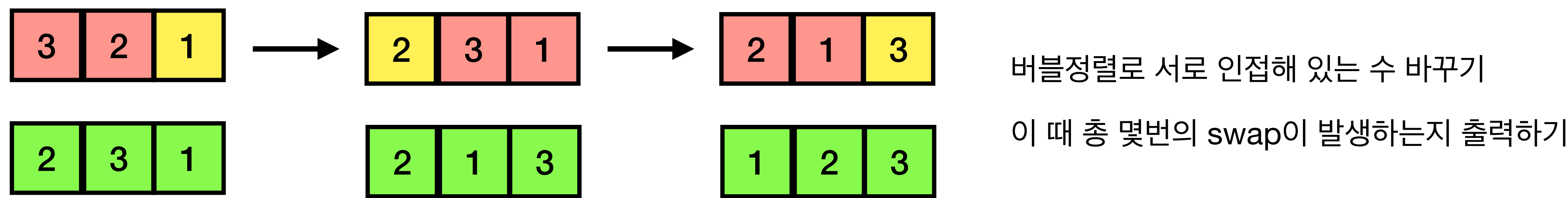


문제 21번

말만 버블정렬이지 실제로는 병합 정렬 과정에서 일어나는 swap의 횟수를 counting하는 문제이다.

n의 최대 범위가 1,000,000이므로 병합정렬을 사용해야 한다.

병합정렬 자체는 동일하게 진행되나 특정 조건에서 swap의 횟수를 result에 계속 더해주면 된다.



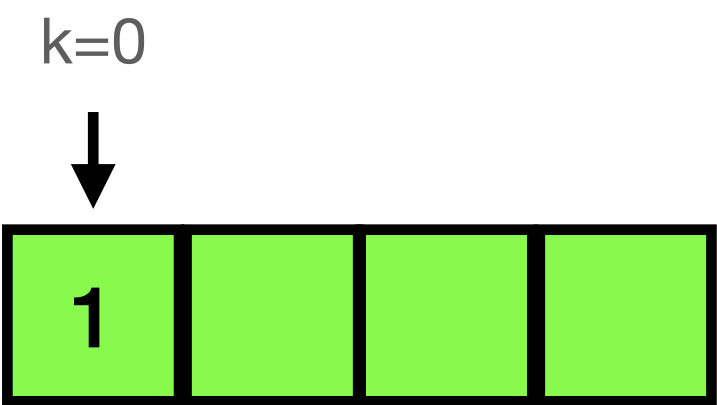
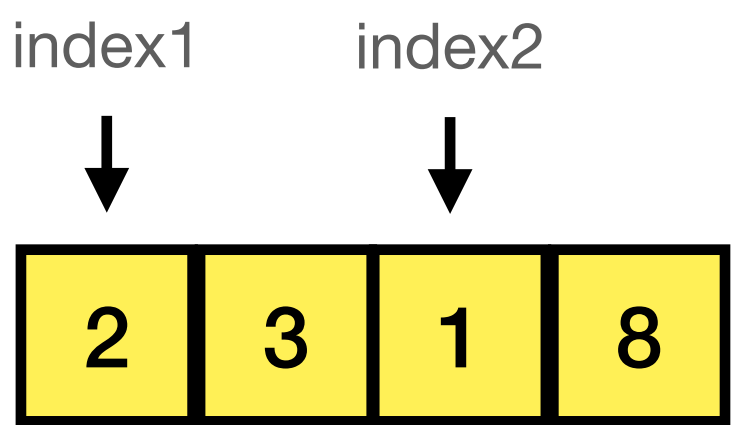
`index2`가 가리키고 있는 값 [5]는 [24,32,42,60]보다 모두 작으므로 계속 앞으로 이동한다.

그러므로 swap이 4번 일어난다.

문제 21번

index1이 가리키는 값 [2]가 index2가 가리키는 값 [1]보다 더 큰 경우 swap이 일어난다.

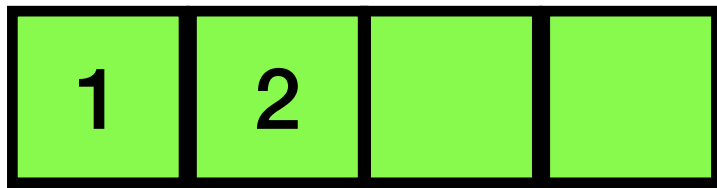
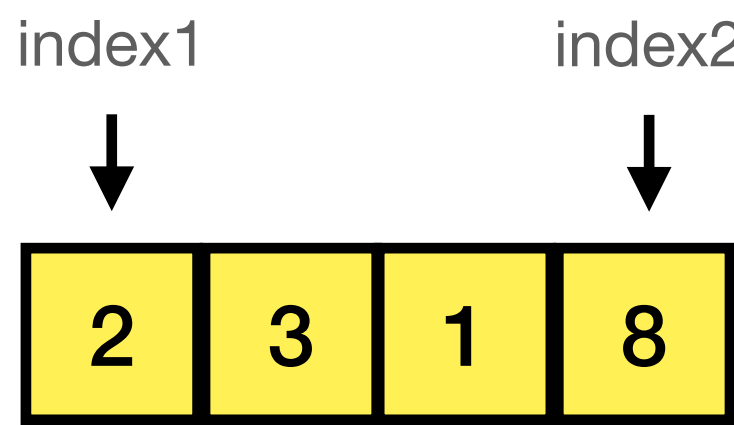
result = result + index2 - K



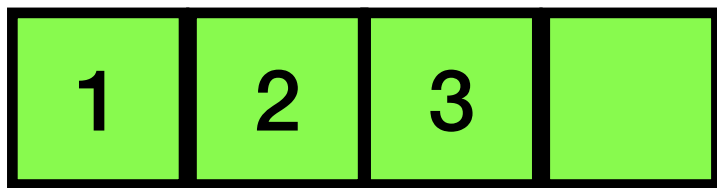
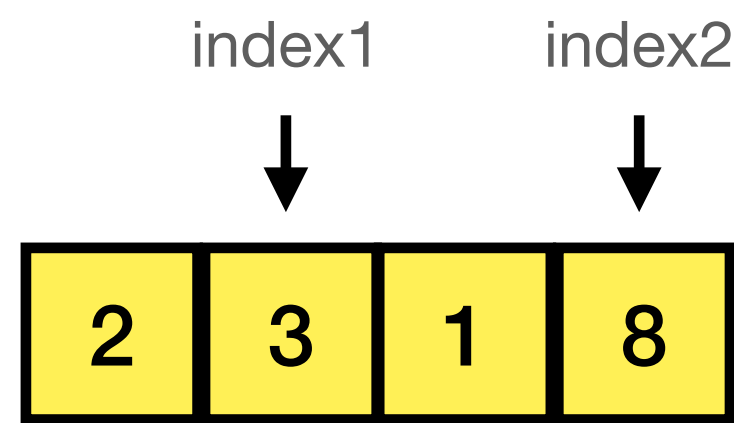
2 > 1 swap 발생

result = 0 index2 = 2 k = 0

result = 2



2 < 8

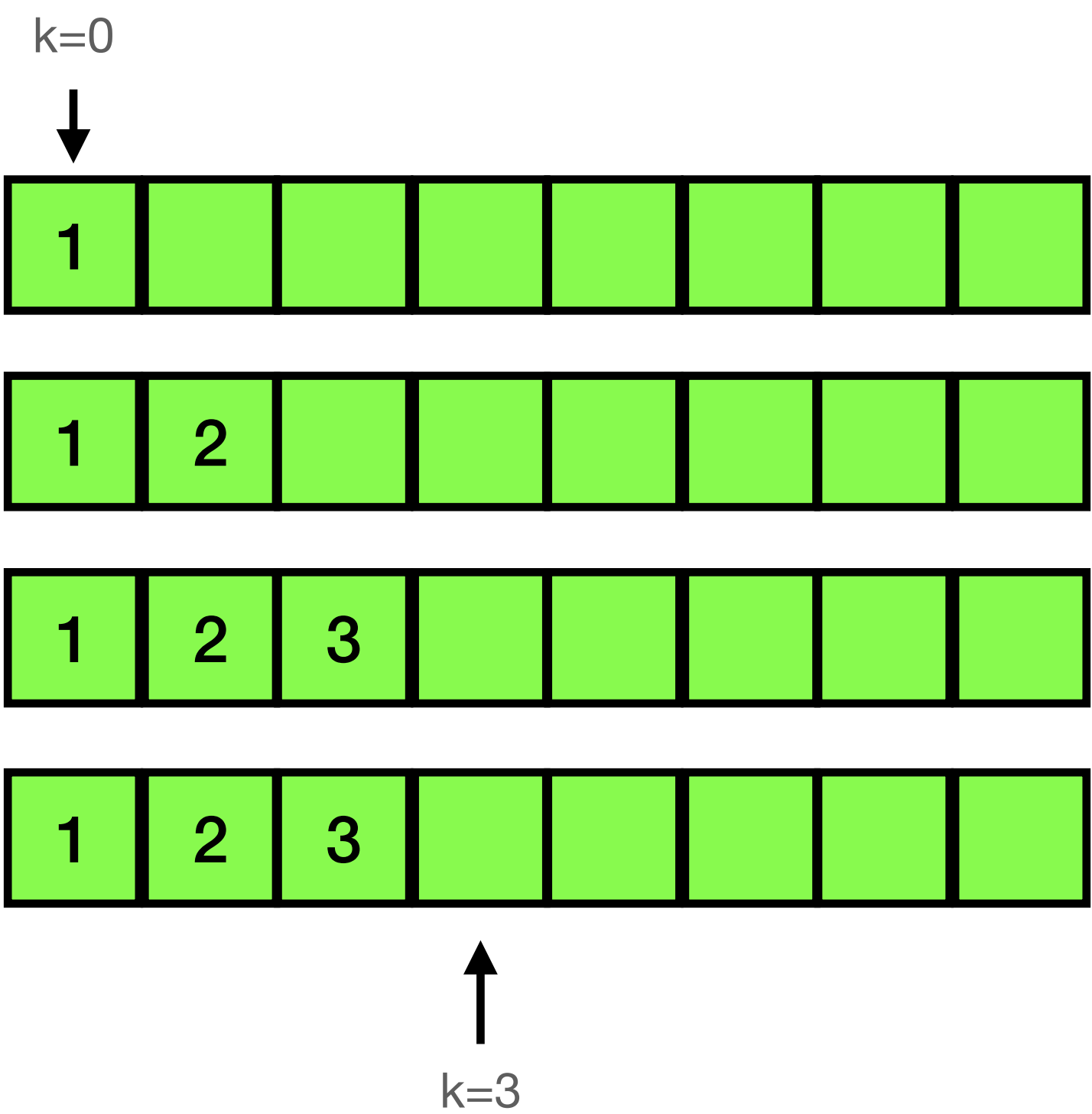
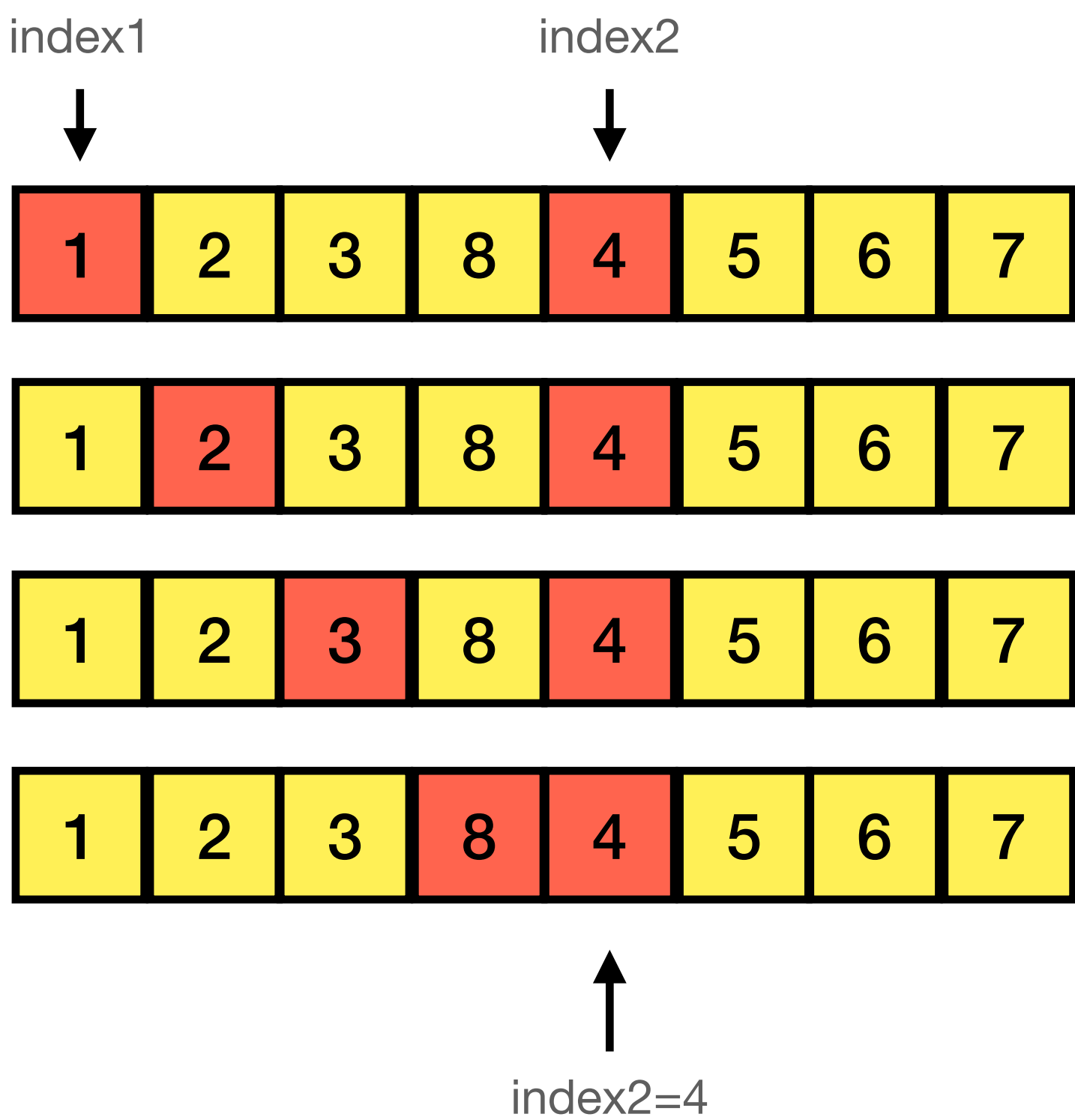


3 < 8

문제 21번

index1이 가리키는 값 [2]가 index2가 가리키는 값 [1]보다 더 큰 경우 swap이 일어난다.

result = result + index2 - K



8 > 4 swap 발생

result = x index2 = 4 k = 3

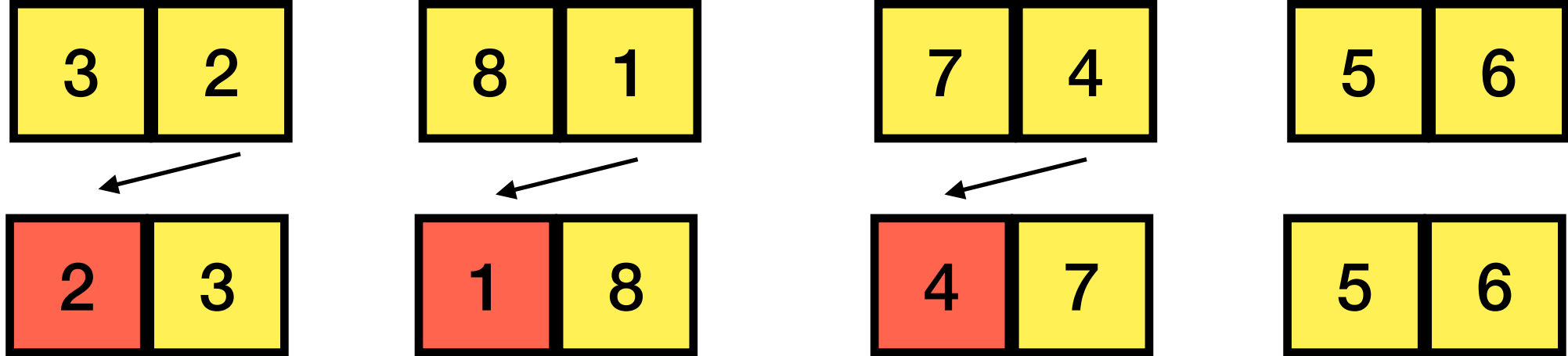
result = x + 1

문제 21번

병합 과정에서 swap횟수를 result에 더해주기

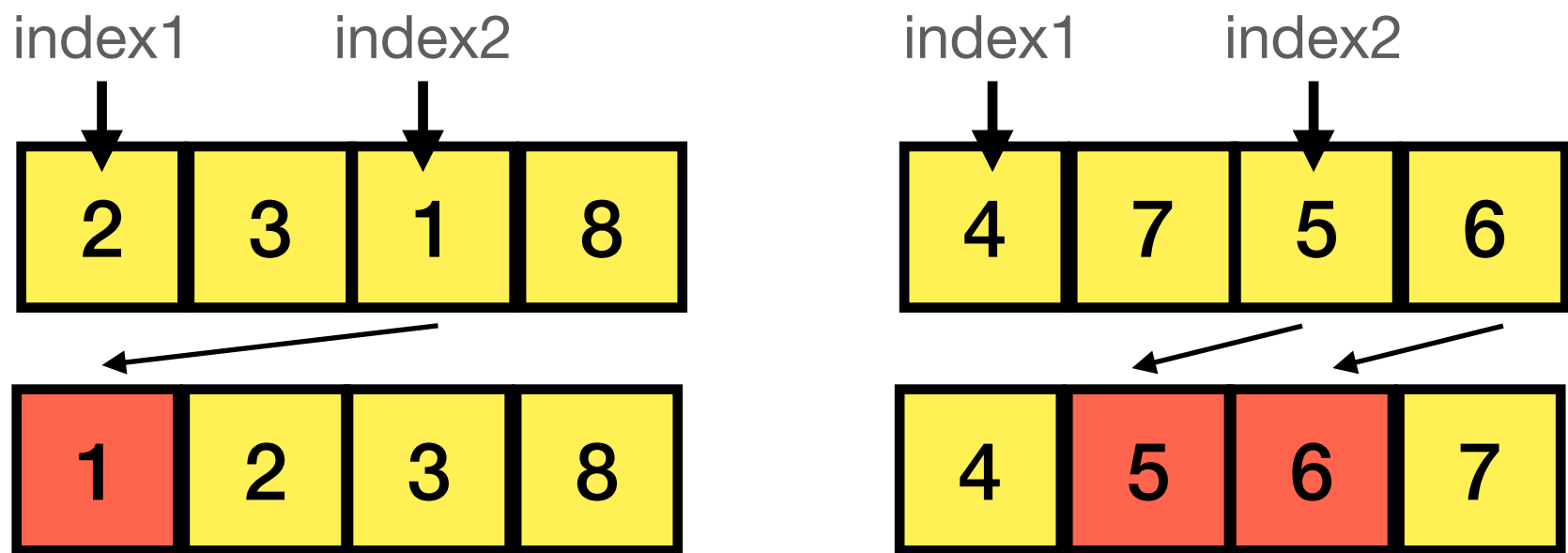
처음 result = 0으로 세팅

STEP1



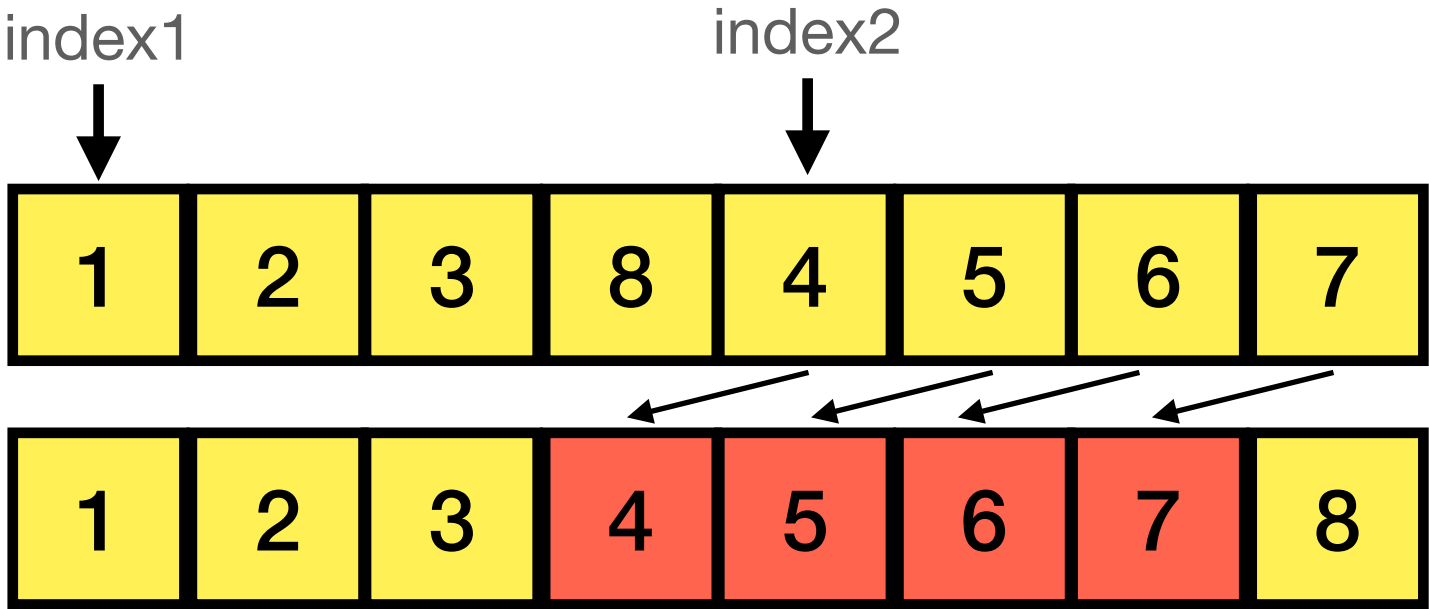
result +=3

STEP2



result +=4

STEP3



result +=4

문제 21번

슈도코드

로직을 수행하면서 뒤쪽 데이터값이 더 작아 선택될 때
swap이 일어난 것과 동일한 것이기 때문에
현재 남은 앞쪽 그룹 데이터의 개수만큼 결괏값에 더해 줌

실제코드

```
result = 0
```

```
while index1 <=m and index2 <=e:
```

```
    if tmp[index1] > tmp[index2]:
```

```
        result = result + index2 - k
```