SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 응용

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

카운팅 정렬

SAMSUNG SW ACADEMY FOR YOUTH

목차

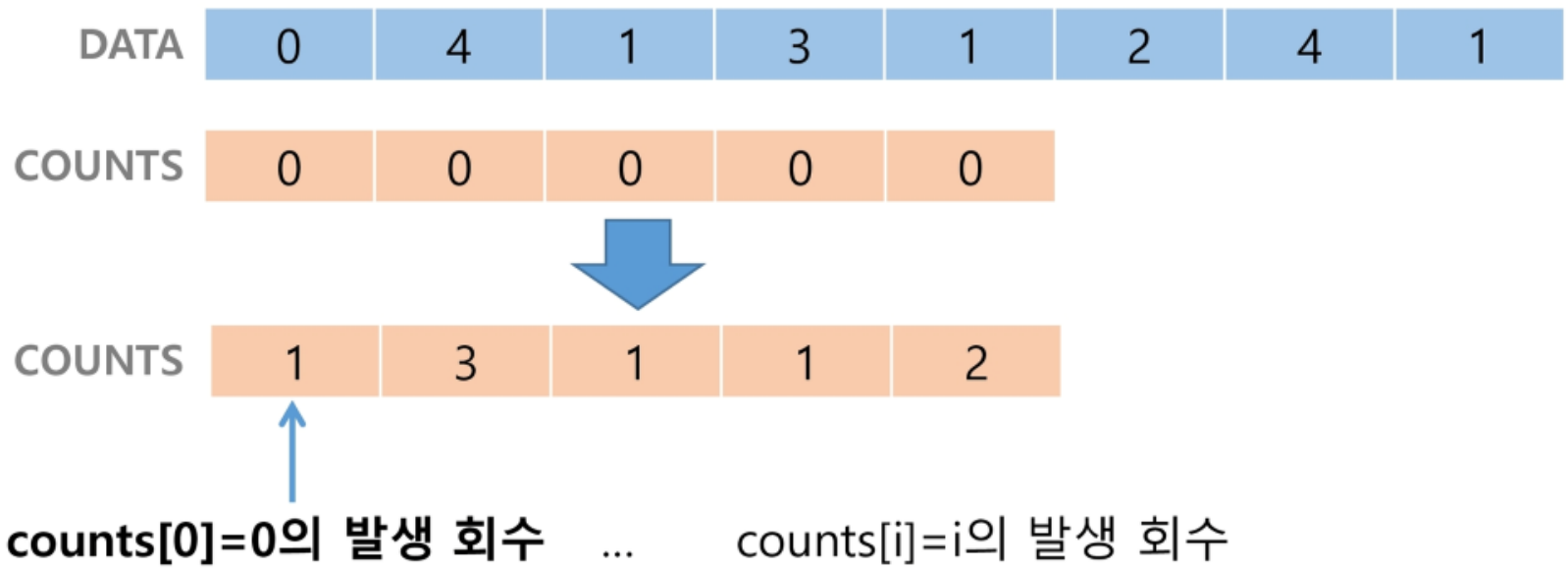
카운팅 정렬(Counting Sort)

- **항목들의 순서를 결정하기 위해 집합에 각 항목이 몇 개씩 있는지 세는 작업을 하여, 선형 시간에 정렬하는 효율적인 알고리즘**
- **제한 사항**
  - 정수나 정수로 표현할 수 있는 자료에 대해서만 적용 가능
    - ✓ 각 항목의 발생 회수를 기록하기 위해, 정수 항목으로 인덱스 되는 카운트들의 배열을 사용하기 때문이다.
  - 카운트들을 위한 충분한 공간을 할당하려면 집합 내의 가장 큰 정수를 알아야 한다.
- **시간 복잡도**
  - O(n + k) : n은 리스트 길이, k는 정수의 최대값

카운팅 정렬(Counting Sort) 과정

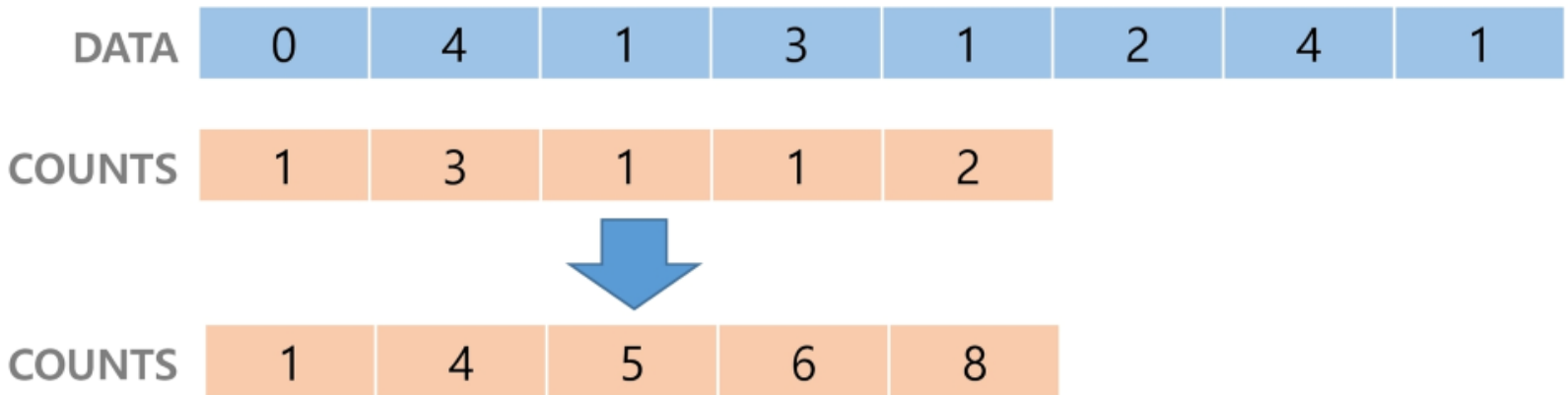
- **{0, 4, 1, 3, 1, 2, 4, 1} 을 카운팅 정렬하는 과정**

- **1단계**
  - Data에서 각 항목들의 발생 회수를 세고, 정수 항목들로 직접 인덱스 되는 카운트 배열 counts에 저장한다.



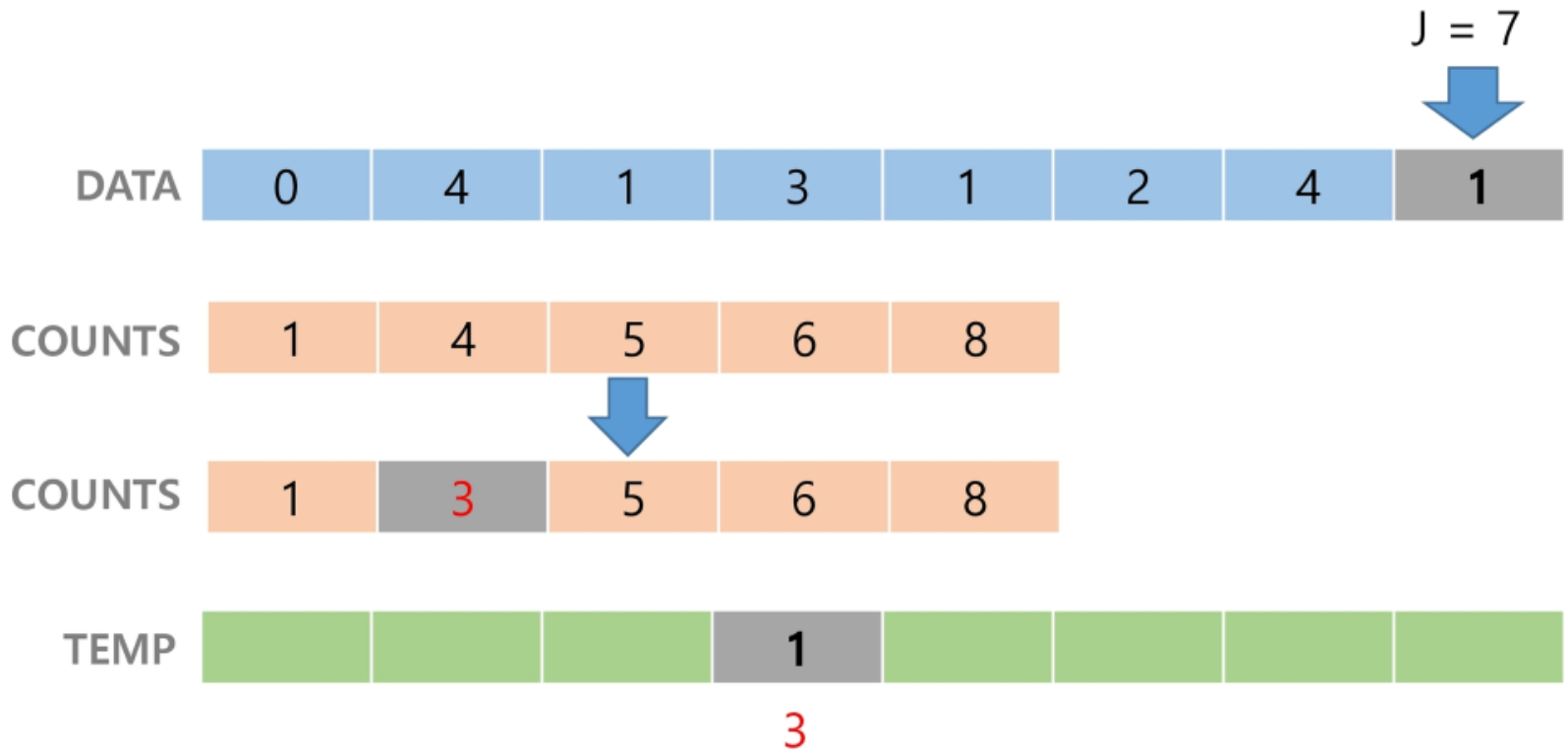
카운팅 정렬(Counting Sort) 과정

- 정렬된 집합에서 각 항목의 앞에 위치할 항목의 개수를 반영하기 위해 counts의 원소를 조정한다.



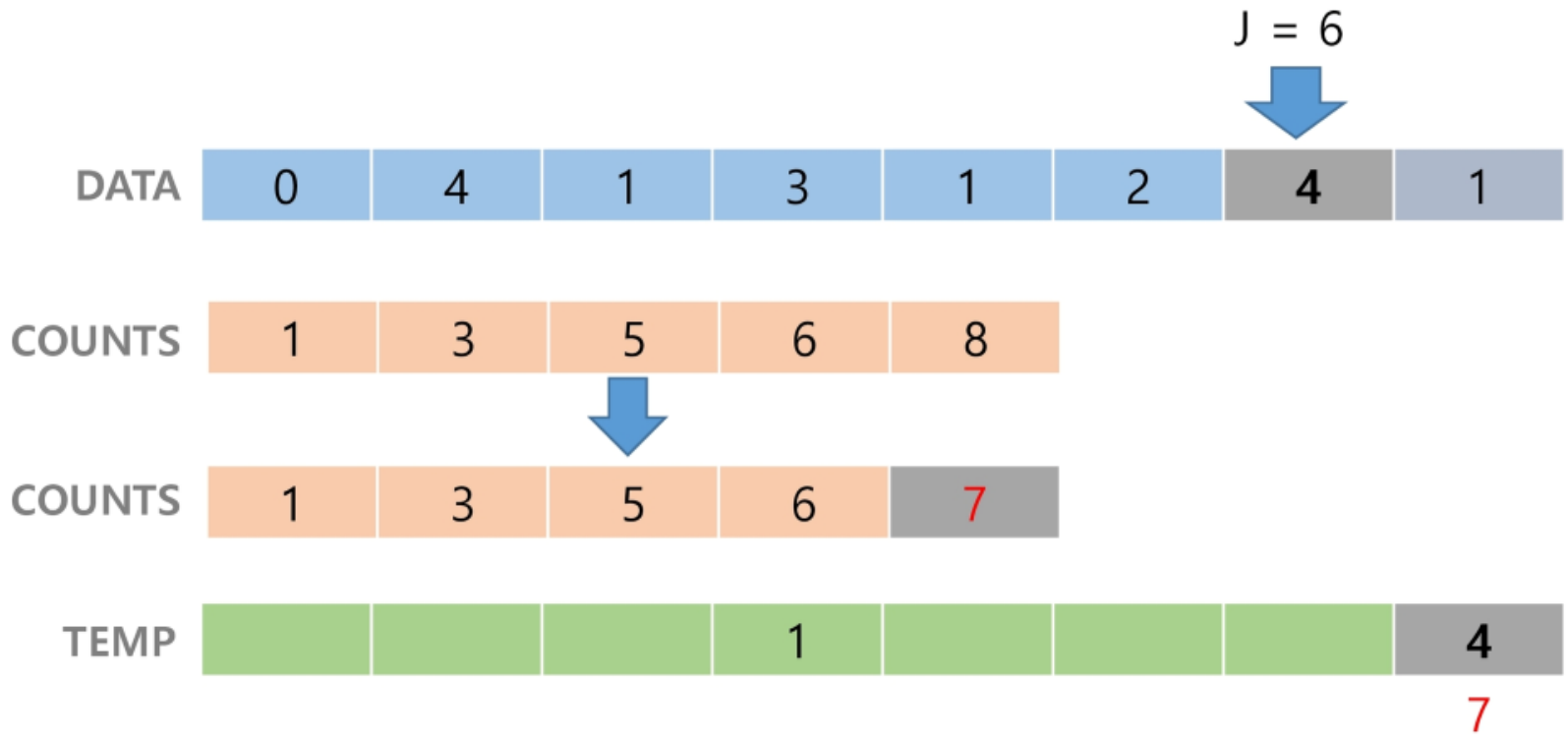
카운팅 정렬(Counting Sort) 과정

- counts[1]을 감소시키고 Temp[3]에 1을 저장한다.



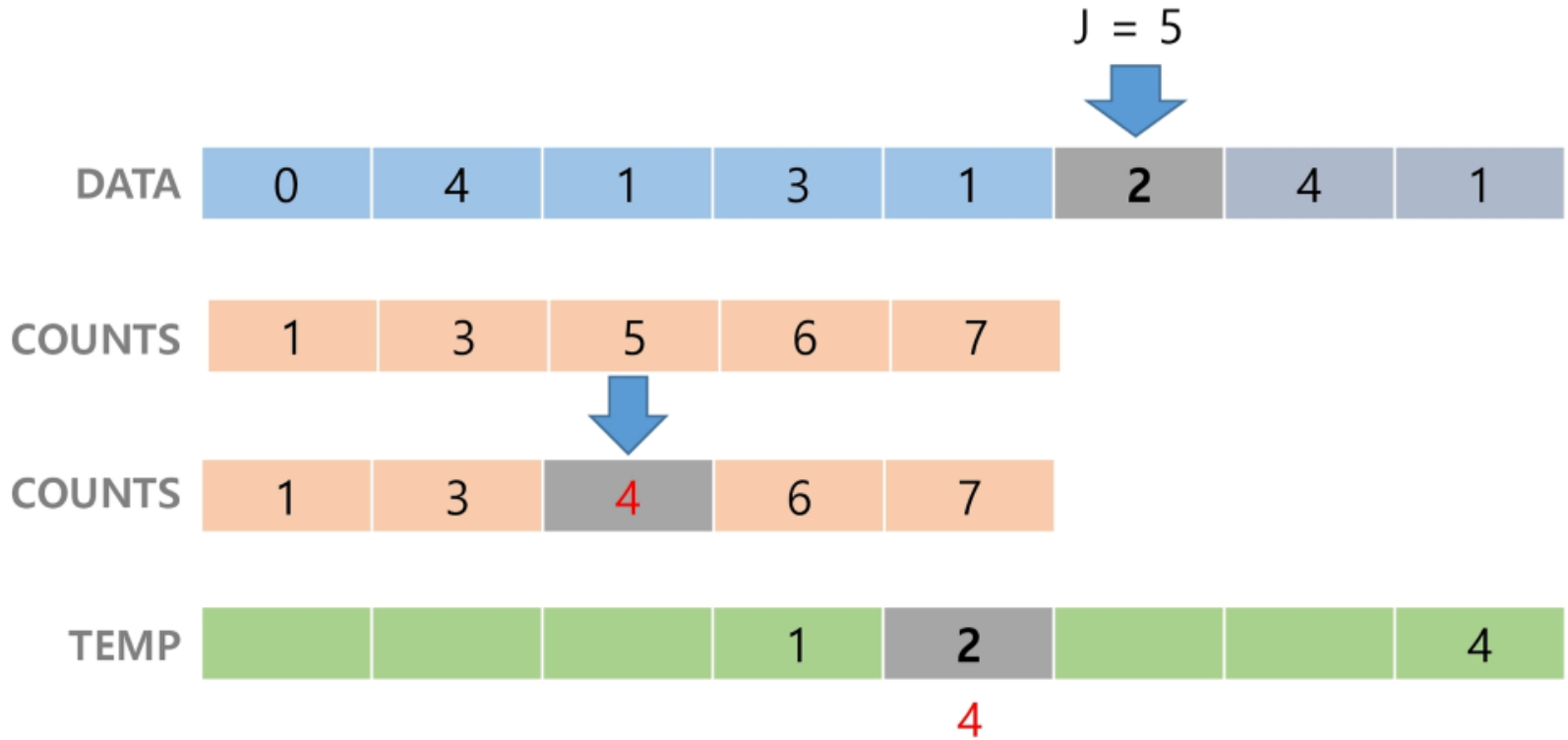
카운팅 정렬(Counting Sort) 과정

- counts[4]를 감소시키고 Temp[7] 에 4를 저장한다.



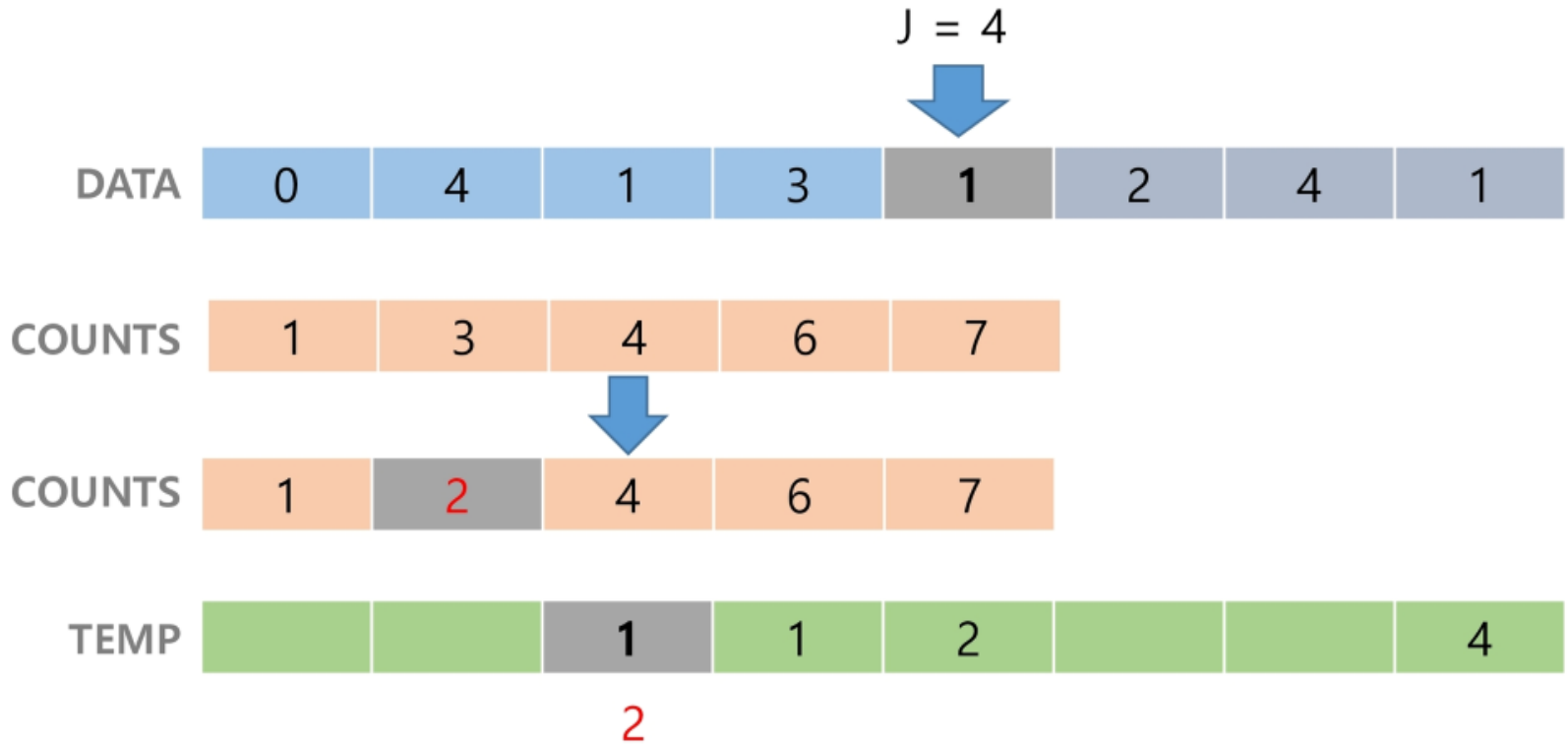
카운팅 정렬(Counting Sort) 과정

- counts[2]를 감소시키고 Temp[4] 에 2를 저장한다.



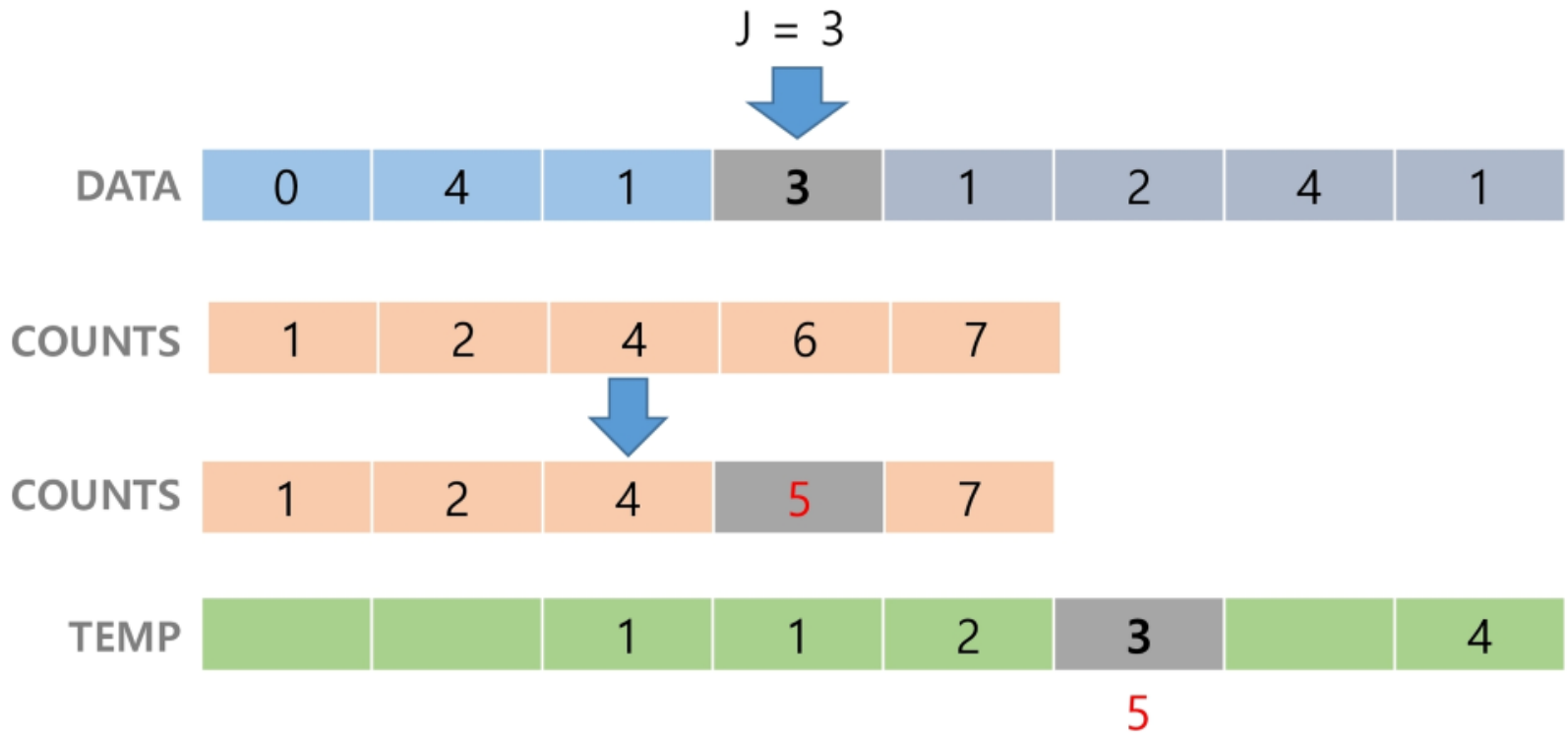
카운팅 정렬(Counting Sort) 과정

- counts[1]을 감소시키고 Temp[2]에 1을 저장한다.



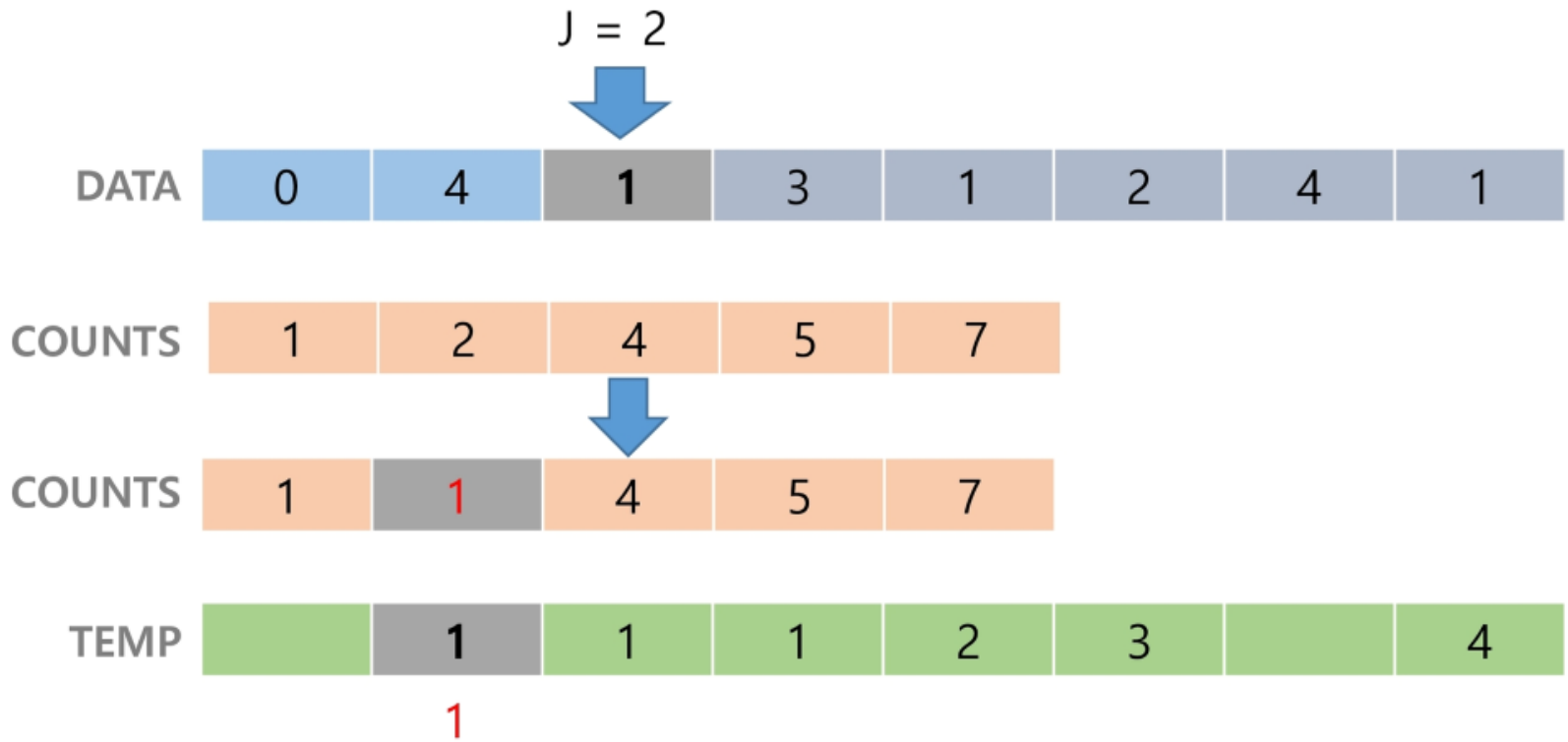
카운팅 정렬(Counting Sort) 과정

- counts[3]을 감소시키고 Temp[5]에 3을 저장한다.



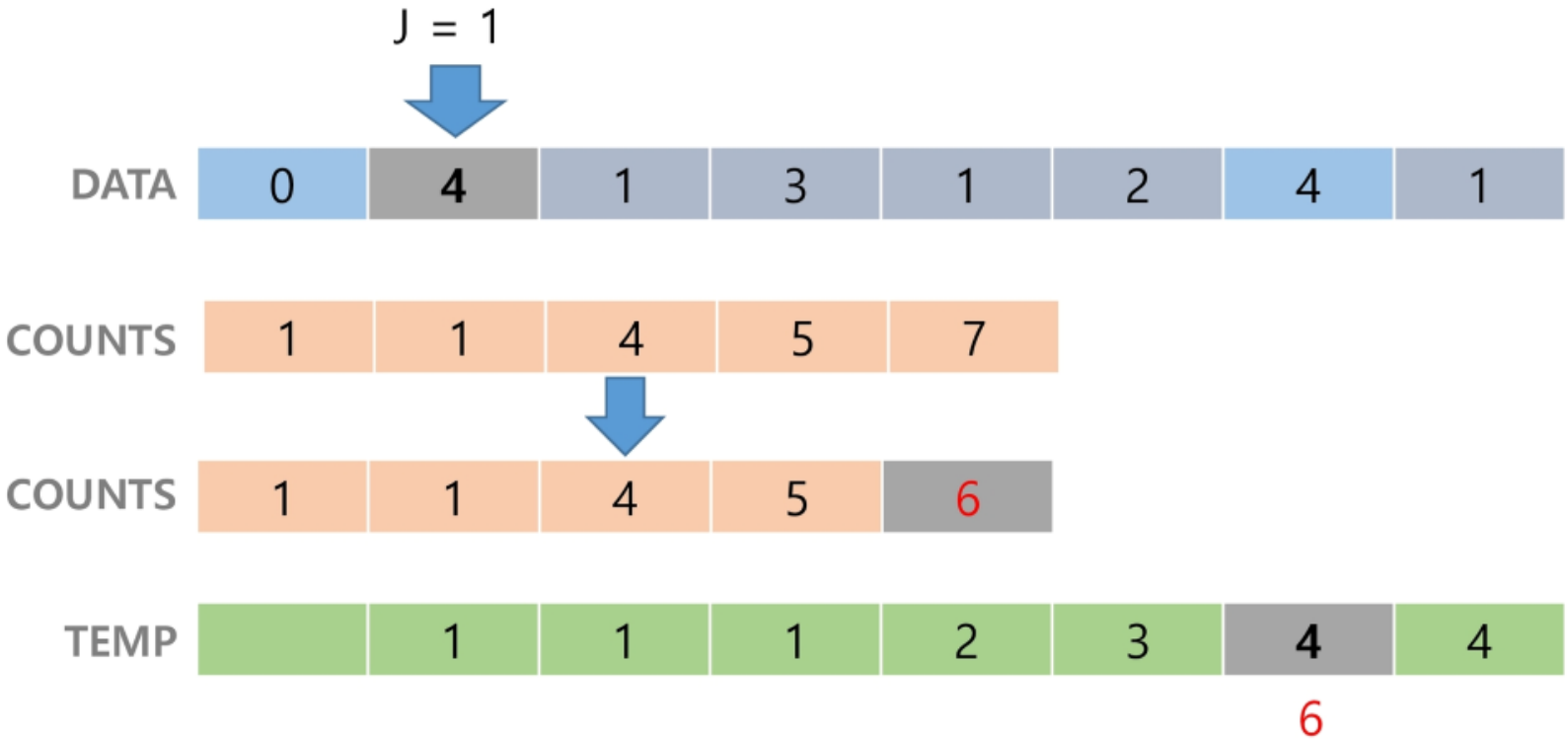
카운팅 정렬(Counting Sort) 과정

- counts[1]을 감소시키고 Temp[1]에 1을 저장한다.



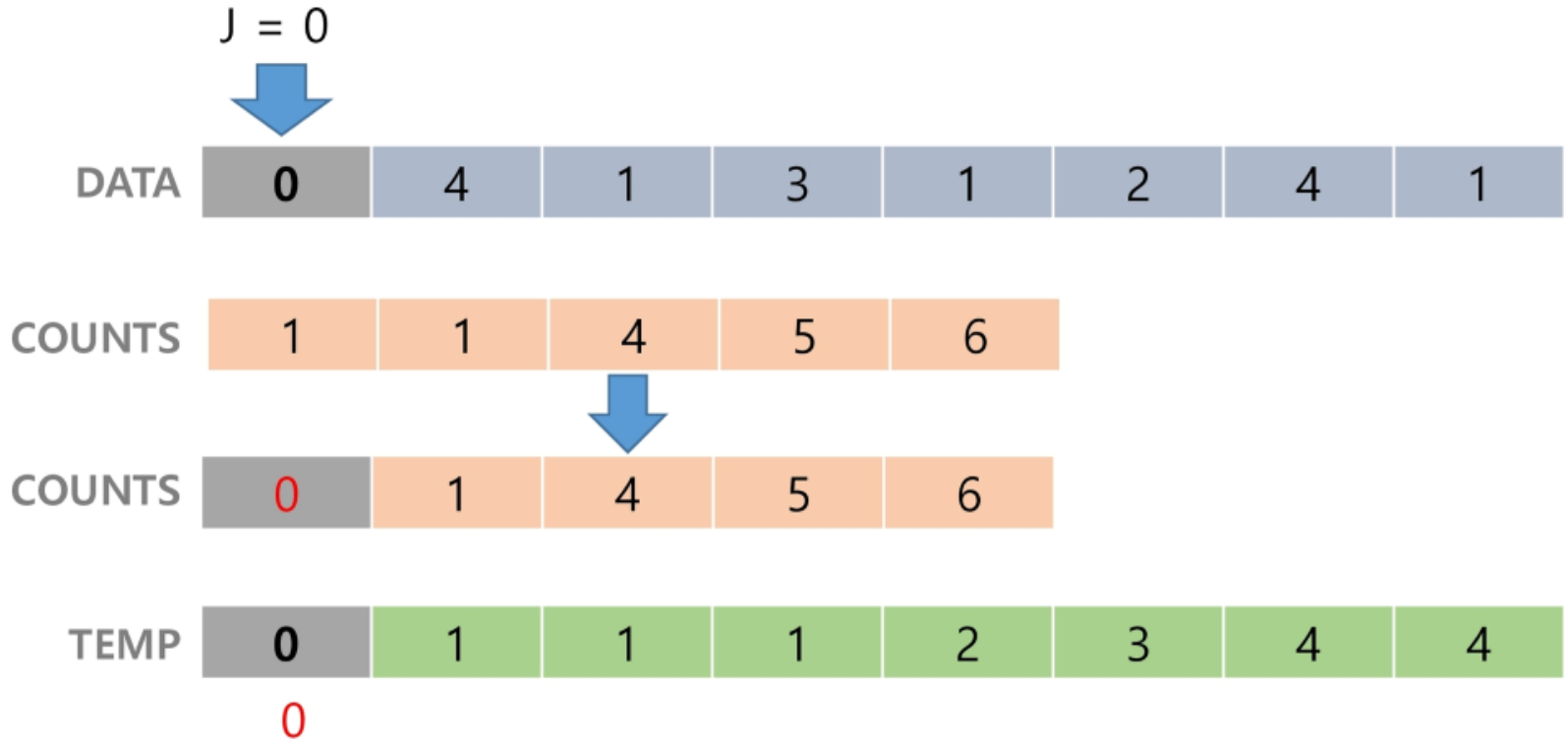
카운팅 정렬(Counting Sort) 과정

- counts[4]를 감소시키고 Temp[6]에 4를 저장한다.



카운팅 정렬(Counting Sort) 과정

- counts[0]를 감소시키고 Temp[0]에 0을 저장한다.



- 정렬 작업을 종료한다.

카운팅 정렬(Counting Sort) 알고리즘

알고리즘

```
countingSort(A, B, k)    // k : 배열원소 중 최대값
// A [1 .. n] -- 입력 배열(값은 1 to k), B [1 .. N] - 정렬된 배열, C [1 .. K] - 카운트 배열
    FOR i = 1 to k do
        C[i] = 0
    FOR j = 1 to n do
        C[A[j]] = C[A[j]]+1
    FOR i = 2 to k do
        C[i] = C[i] + C[i-1]
    FOR j = n to 1 do
        B[--C[A[j]]] = A[j]
end countingSort
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

선택 정렬

선택 정렬(Selection Sort)

포켓볼 순서대로 정렬하기

- 왼쪽과 같이 흩어진 당구공을 오른쪽 그림처럼 정리한다고 하자. 어떻게 하겠는가?

- 많은 사람들은 당구대 위에 있는 공 중 가장 작은 숫자의 공부터 골라서 차례대로 정리할 것이다. 이것이 바로 선택 정렬이다.

선택 정렬(Selection Sort)

- **주어진 자료들 중 가장 작은 값의 원소부터 차례대로 선택하여 위치를 교환하는 방식**
  - 앞서 살펴본 셀렉션 알고리즘을 전체 자료에 적용한 것이다.

- **정렬 과정**
  - 주어진 리스트 중에서 최소값을 찾는다.
  - 그 값을 리스트의 맨 앞에 위치한 값과 교환한다.
  - 맨 처음 위치를 제외한 나머지 리스트를 대상으로 위의 과정을 반복한다.

- **시간 복잡도**
  - $O(n^2)$

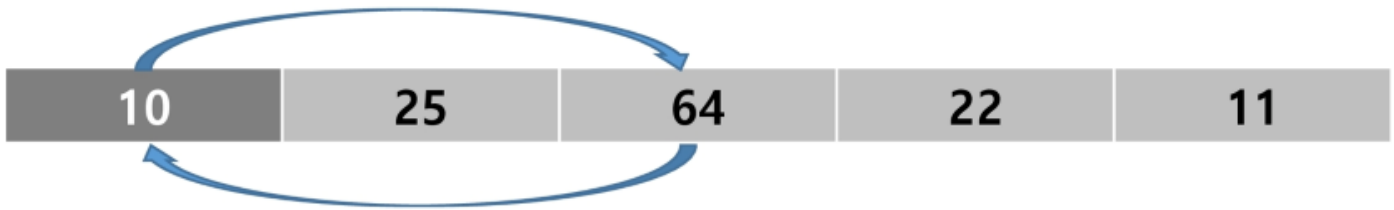
선택 정렬(Selection Sort) 과정

정렬 과정

정렬된 원소 | 미정렬 원소

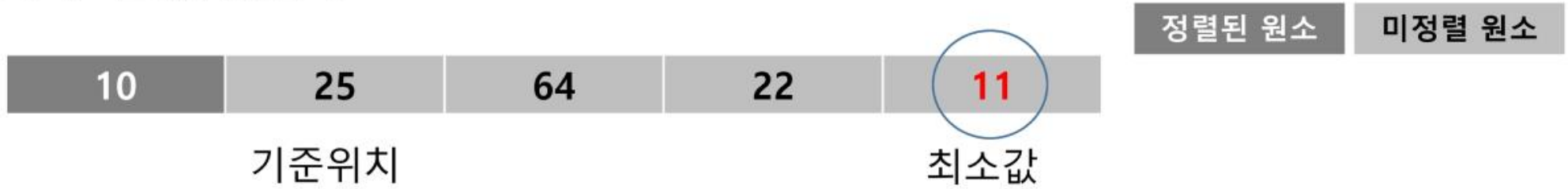
① 주어진 리스트에서 최소값을 찾는다.

② 리스트의 맨 앞에 위치한 값과 교환한다.

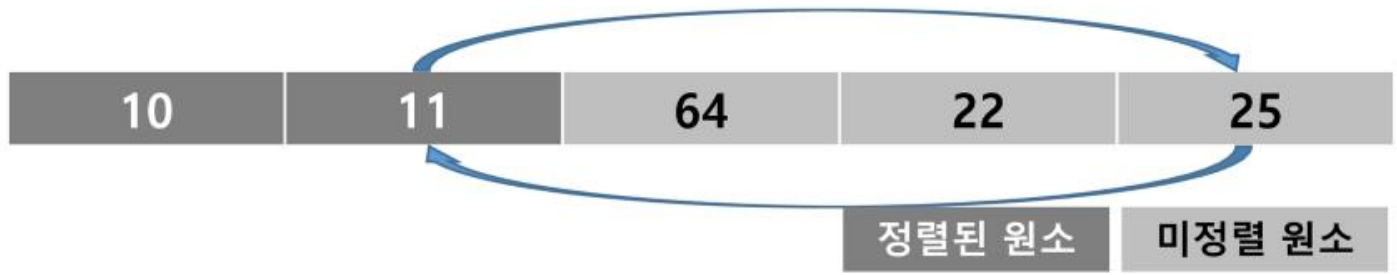


선택 정렬(Selection Sort) 과정

③ 미정렬 리스트에서 최소값을 찾는다.



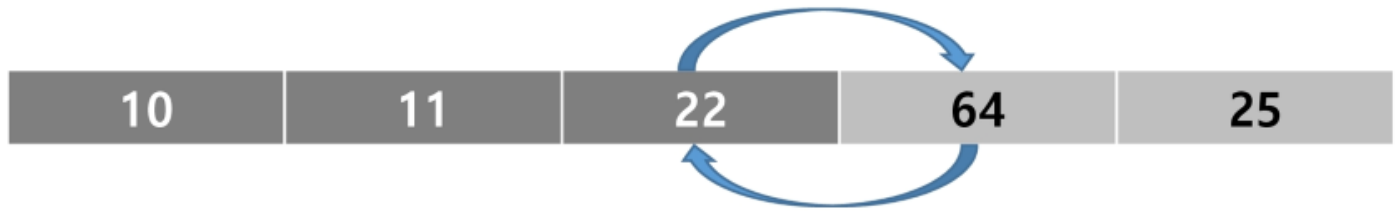
④ 리스트의 맨 앞에 위치한 값과 교환한다.



선택 정렬(Selection Sort) 과정

⑤ 미정렬 리스트에서 최소값을 찾는다.

⑥ 리스트의 맨 앞에 위치한 값과 교환한다.



선택 정렬(Selection Sort) 과정

⑦ 미정렬 리스트에서 최소값을 찾는다.

⑧ 리스트의 맨 앞에 위치한 값과 교환한다.

▪ 미정렬원소가 하나 남은 상황에서는 마지막 원소가 가장 큰 값을 갖게 되므로, 실행을 종료하고 선택 정렬이 완료된다.

선택 정렬(Selection Sort) 알고리즘

알고리즘

```
selectionSort(a[], size) // a: 정렬할 배열 , n: 배열의 크기
        i, j, t, min, temp;
        FOR i from 0 to size-2 { // 미정렬 원소의 시작 : 끝원소는 비교대상이 없으므로 수행하지 않음
            min ← i;
            FOR j from i+1 to size-1 {
                IF (a[j]<a[min]) THEN min ← j;
            }
            temp ← a[i];
            a[i] ← a[min];
            a[min] ← temp;
        }
end selectionSort
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

퀵 정렬

퀵 정렬(Quick Sort)

- 주어진 배열을 두 개로 분할하고, 각각을 정렬한다.
- 퀵 정렬은 분할할 때, 기준 아이템(pivot item) 중심으로, 이보다 작은 것은 왼편, 같거나 큰 것은 오른편에 위치시킨다.
- 시간 복잡도
  - O(n log n)

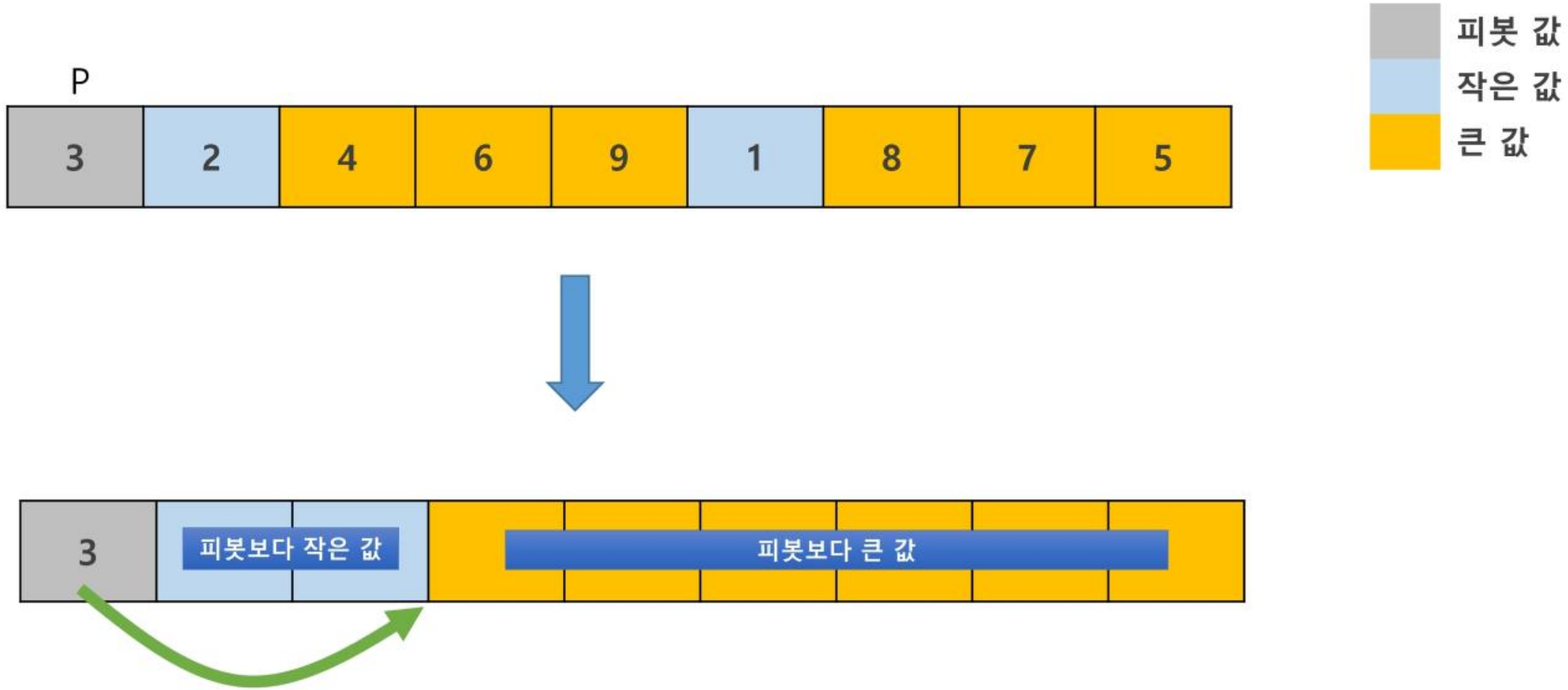
퀵 정렬(Quick Sort)

아이디어

- P(피봇)값들 보다 같거나 큰 값은 오른쪽, 작은 값들은 왼쪽 집합에 위치하도록 한다.

- 피봇을 두 집합의 가운데에 위치시킨다.

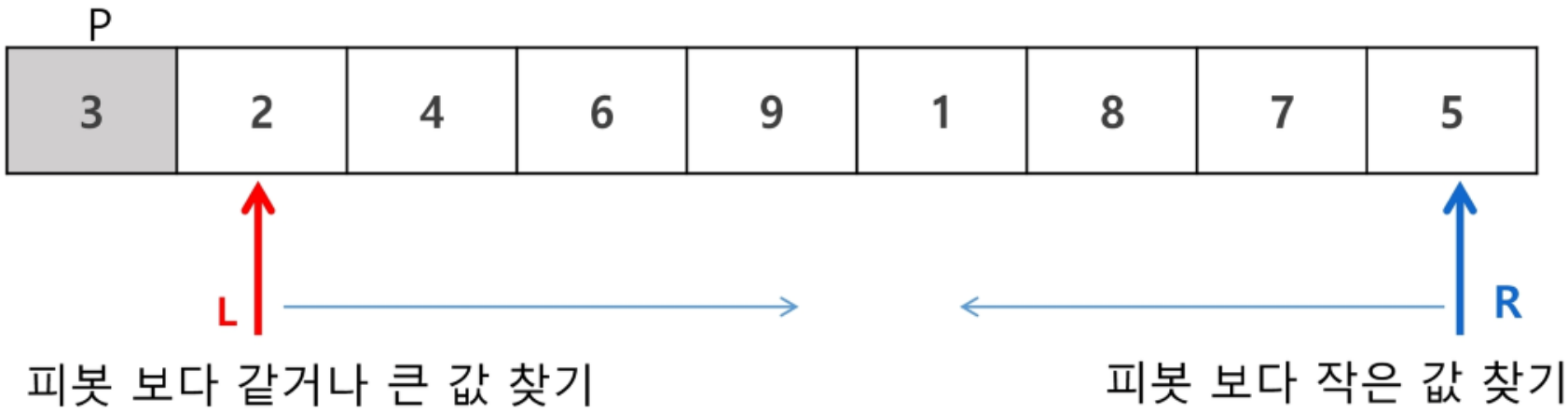
퀵 정렬(Quick Sort)



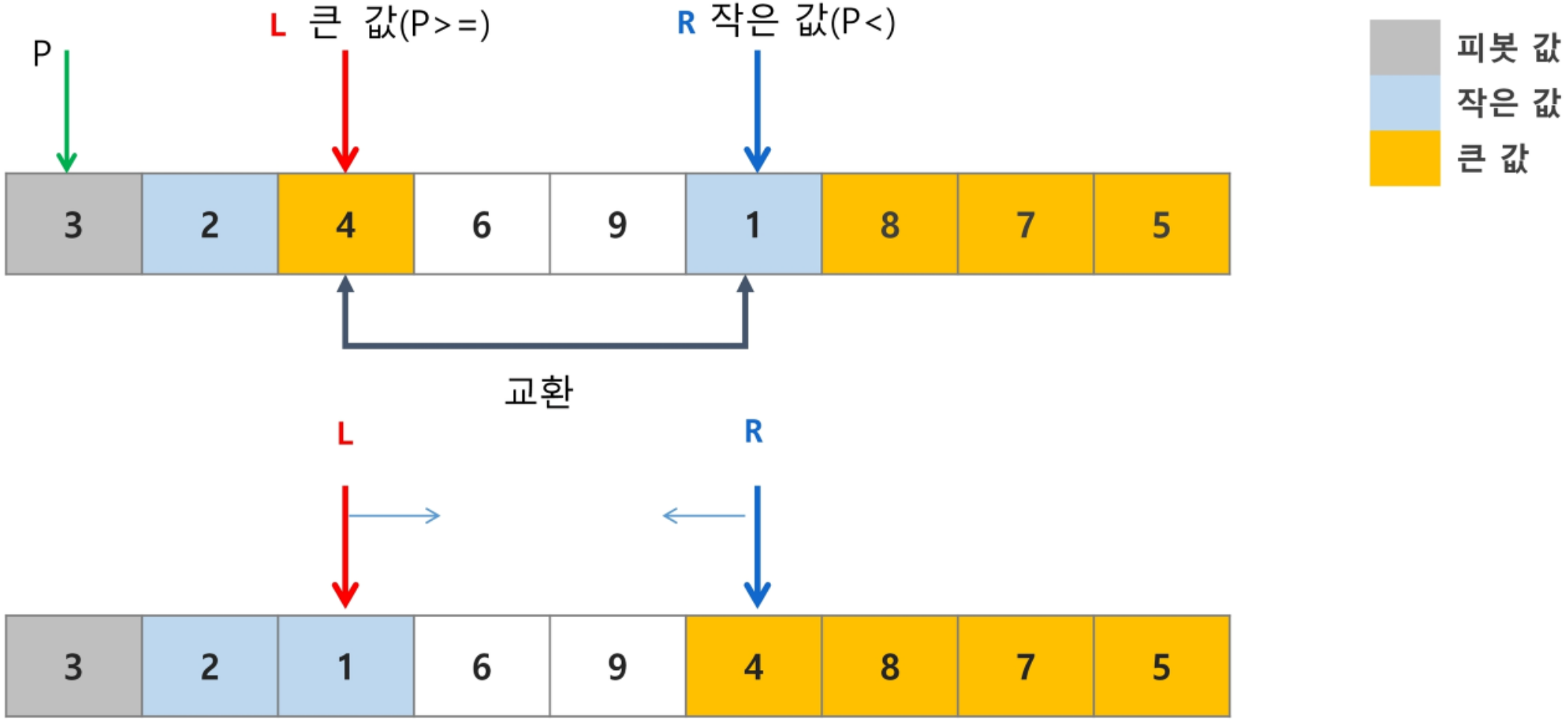
퀵 정렬(Quick Sort)

퀵 정렬(Quick Sort)

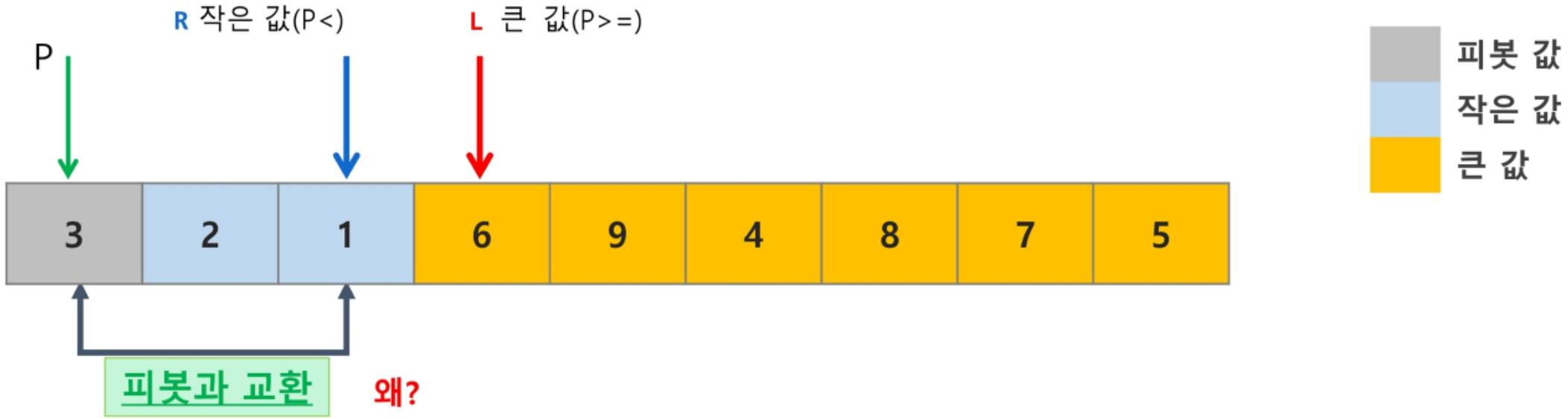
- **피봇 선택**
  - 가장 왼쪽 값



퀵 정렬(Quick Sort) 과정



퀵 정렬(Quick Sort) 과정



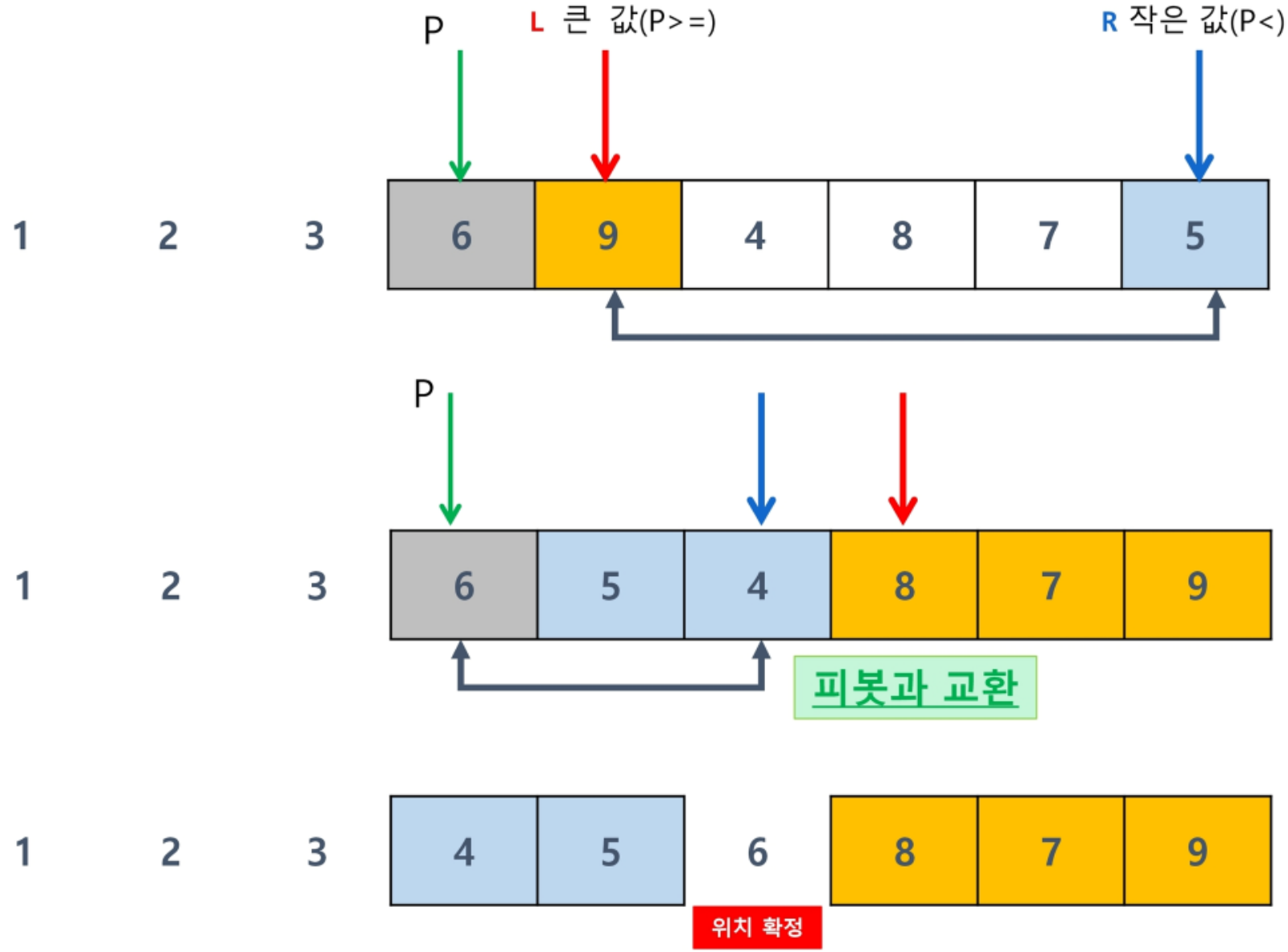
왜?

L와 R이 교차하면 L, R은 피봇을 기준으로작은 값과 큰 값들의 경계에 위치한다.

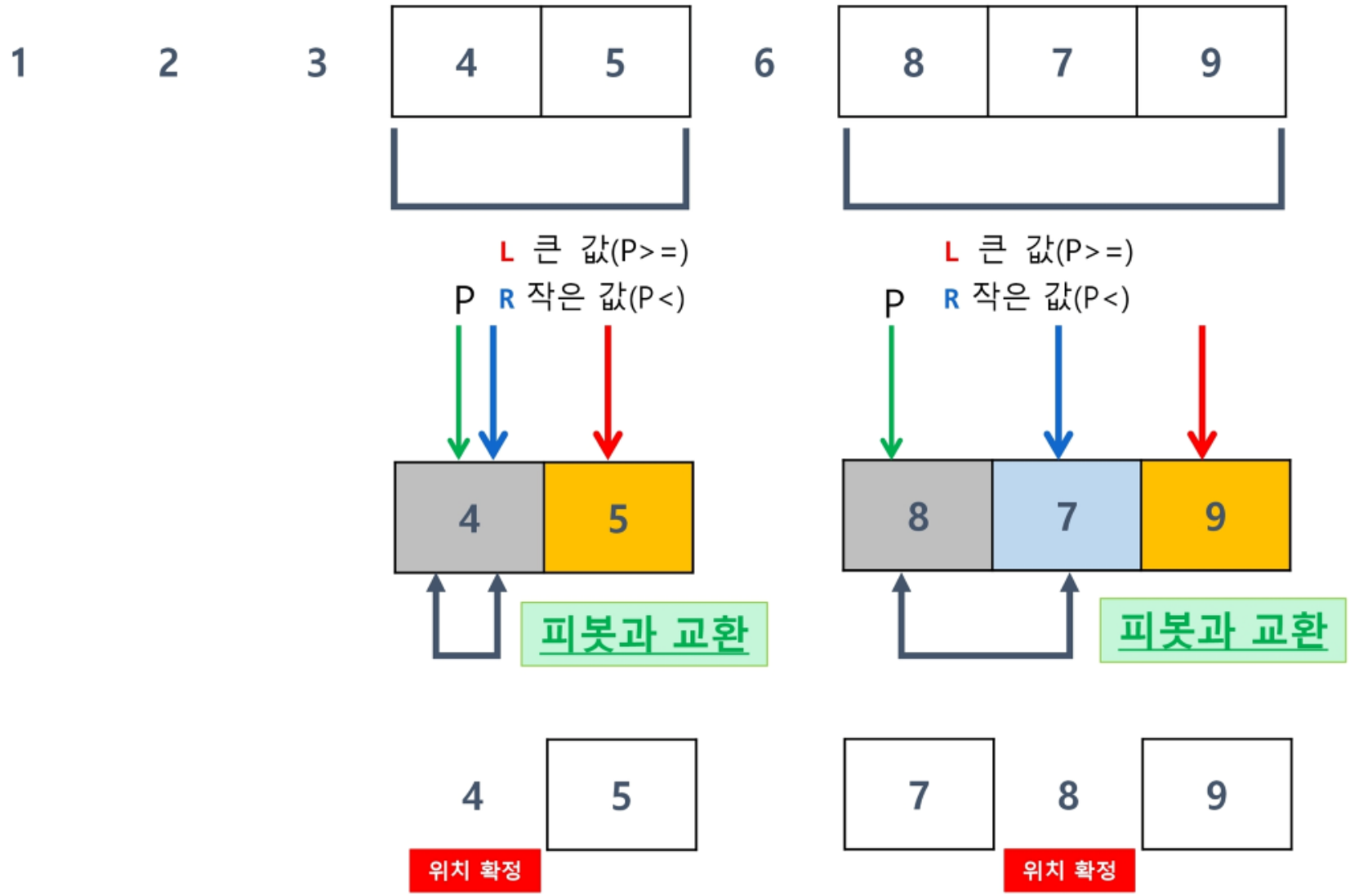
퀵 정렬(Quick Sort) 과정

퀵 정렬(Quick Sort) 과정

퀵 정렬(Quick Sort) 과정



퀵 정렬(Quick Sort) 과정



크기가 1인 집합은 정렬이 필요 없음 ➔ 정렬 끝

퀵 정렬(Quick Sort) 알고리즘

알고리즘

```
quickSort(A[], s, e)
        IF s < e  THEN {
            // p: 피봇 위치
            p ← partition(a, s, e)
            quickSort(A[], s, p - 1)
            quickSort(A[], p + 1, e)
        }
end quickSort
```

```
partition(A[], s, e)
        p ← A[s]          // p: 피봇 값
        l ← s+1,  r ← e
        DO
            WHILE l<e and A[l] < : l++
            WHILE r>s and A[r] ≥ p : r--
            IF l < r then swap(A[l], A[r])
        WHILE l < r
        swap(A[s], A[r])
        RETURN r
end partition
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미