정렬 알고리즘

들어가기 전에...

29	63	6	40	51	93	9	43	53	28
90	59	72	88	61	47	65	2	96	62
31	83	20	78	45	42	85	87	76	57
18	77	32	10	99	1	3	14	52	100
66	71	49	55	68	74	97	4	19	34
75	24	7	64	33	81	5	58	17	79
36	26	82	80	86	37	8	21	70	46
23	15	60	44	35	98	56	92	95	89
16	50	39	25	11	48	67	94	91	73
13	84	41	12	22	30	27	54	69	

big-O 표기법이란?

- 알고리즘의 효율성을 표기하기 위해 사용
  - 최고 차항만 남김 / 계수 생략
  - $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$
- $O(1)$
  - 데이터의 크기와 관계없이 실행 시간이 동일 (index 접근 등)
- $O(\log n)$
  - 연산을 한 번 진행하면 다음 연산을 할 데이터의 크기가 절반이 됨 (up-down 게임)
- $O(n)$
  - 데이터의 크기가 증가하면 실행 시간도 선형적으로 증가 (배열 순차 탐색, 카운팅 정렬)
- $O(n \log n)$
  - $O(n)$의 알고리즘과 $O(\log n)$의 알고리즘이 혼합 (병합 정렬, 퀵 정렬)
- $O(n^2)$
  - $O(n)$의 알고리즘이 2번 중첩 (선택 정렬, 삽입 정렬, 버블 정렬)
- $O(2^n)$
  - 주로 재귀적인 형태

이 코드의 시간복잡도는?

```
void solve() {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            ...
        }

        for (int j = 0; j < N; j++) {
            ...
        }
    }
}
```

이 코드의 시간복잡도는?

```java
// O(log N)
void somethingLog() {
    ...
}

void solve() {
    for (int i = 0; i < N; i++) {
        somethingLog();
    }
    for (int i = 0; i < N; i++) {
        ...
    }
}
```

정렬이란?

- 원소들을 특정 조건에 따라 일정한 순서대로 나열하는 것
  - 오름차순 정렬: $a < b$ 이면 a가 b보다 앞에 오도록 나열
  - 내림차순 정렬: $a > b$ 이면 a가 b보다 앞에 오도록 나열
  - 위상 정렬: $a \rightarrow b$ 간선이 있으면 a가 b보다 앞에 오도록 나열

정렬을 하는 이유

- 데이터를 탐색하기 위해
  - 도서관에서 책이 무작위로 배열되어 있다면 원하는 책을 찾기 위해 모든 책을 뒤져봐야 함

정렬 알고리즘 종류

- 선택 정렬 (Selection Sort)
- 삽입 정렬 (Insertion Sort)
- 버블 정렬 (Bubble Sort)
- 병합 정렬 (Merge Sort)
- 퀵 정렬 (Quick Sort)

- .. 이외에도 많은 알고리즘들이 있다

선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

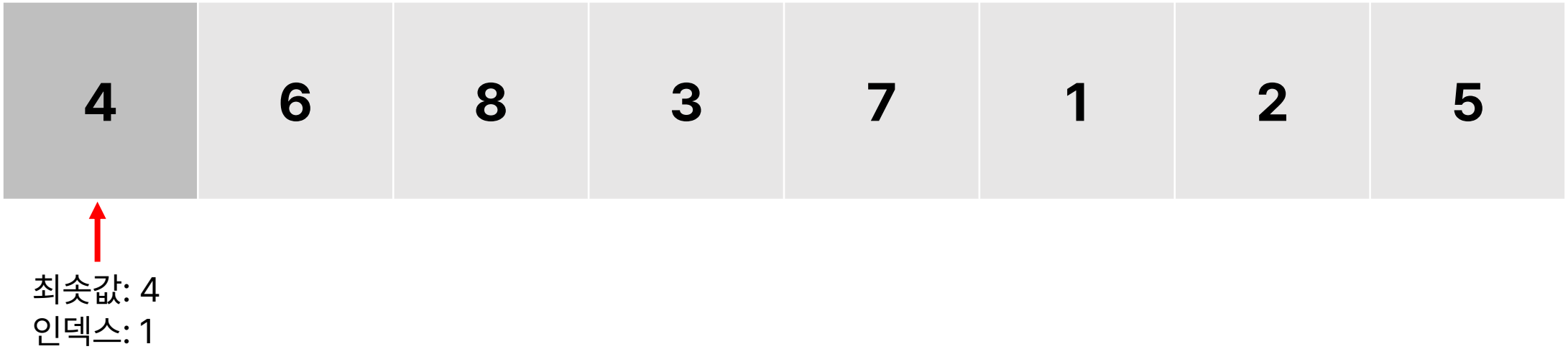
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

4	6	8	3	7	1	2	5

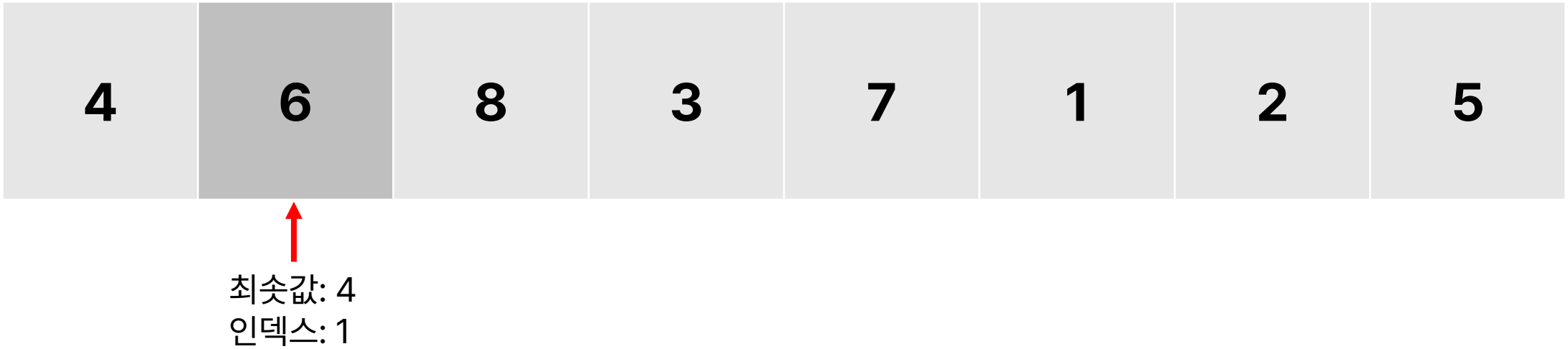
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



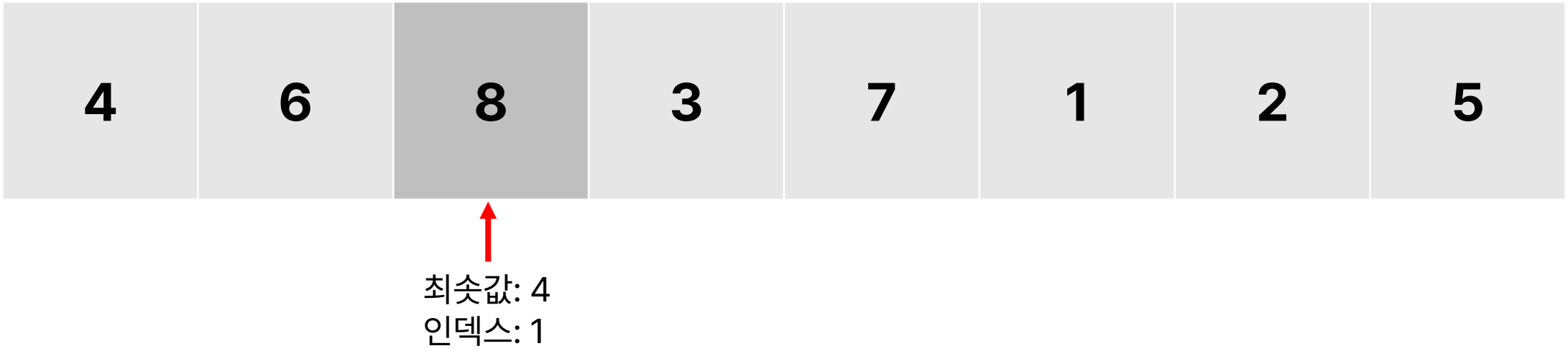
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



선택 정렬 (Selection Sort)

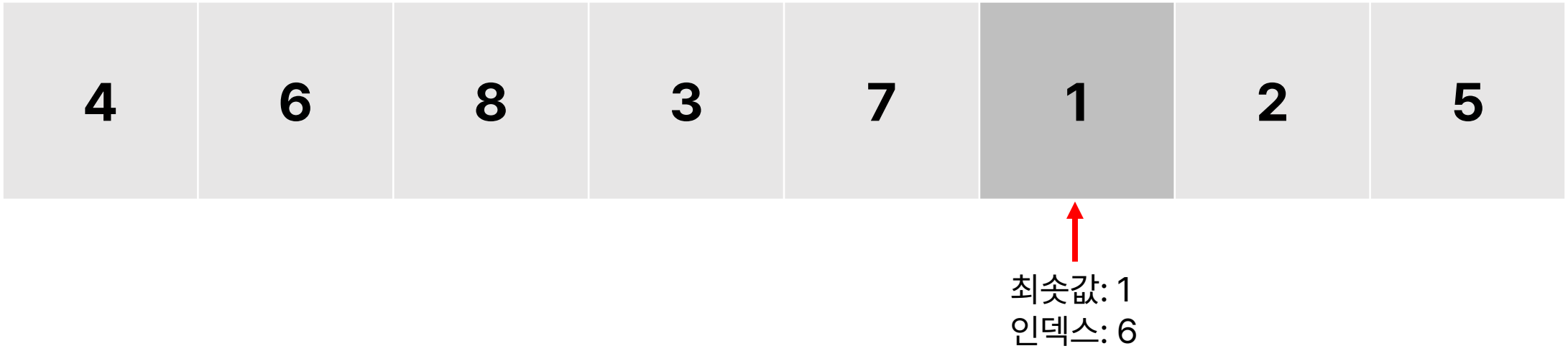
- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

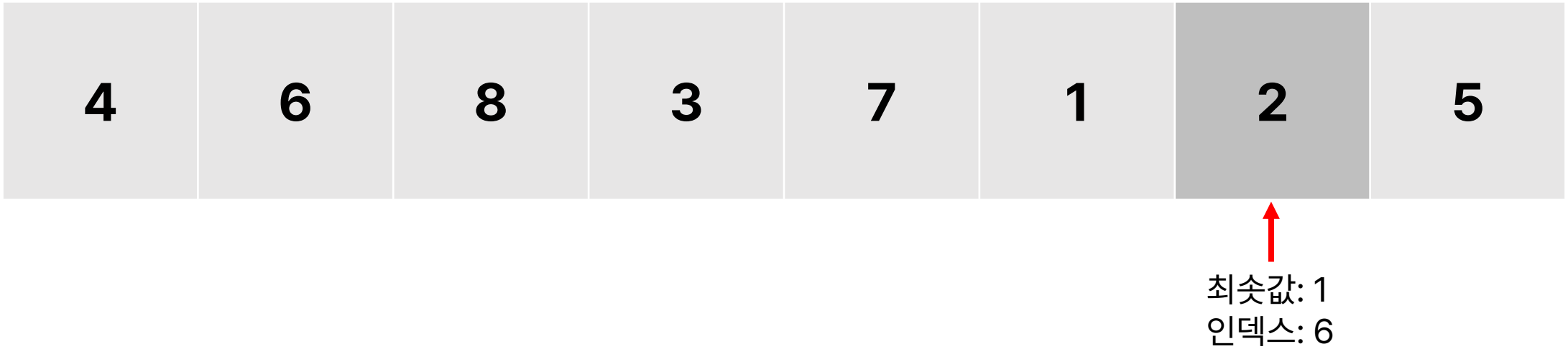
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



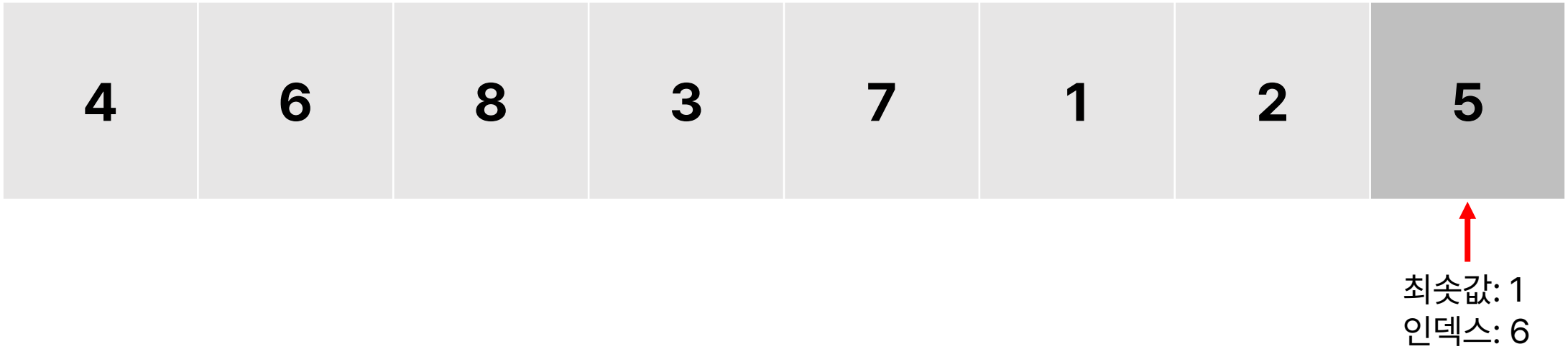
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



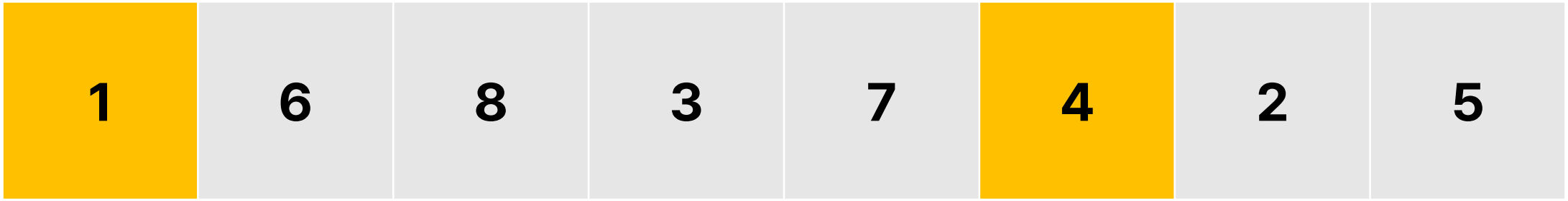
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



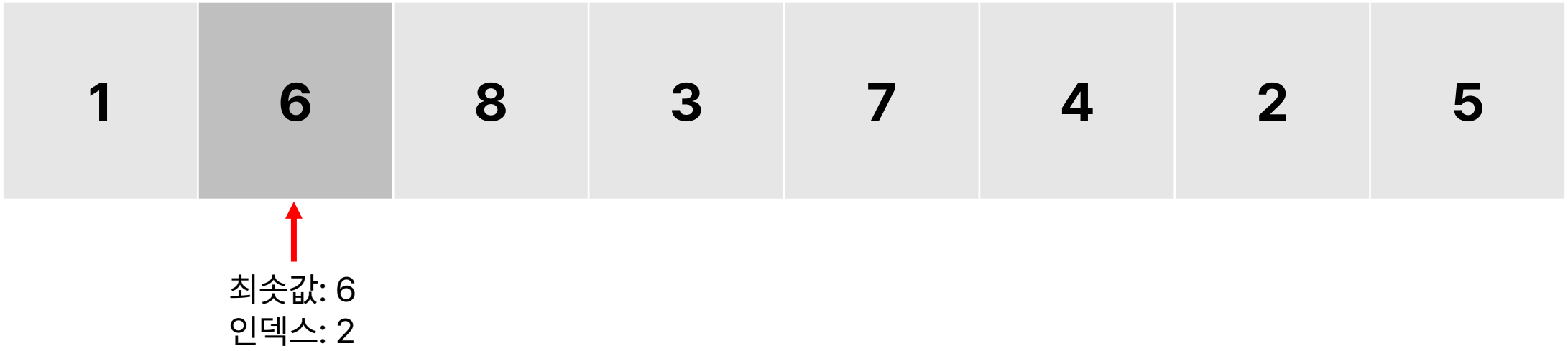
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



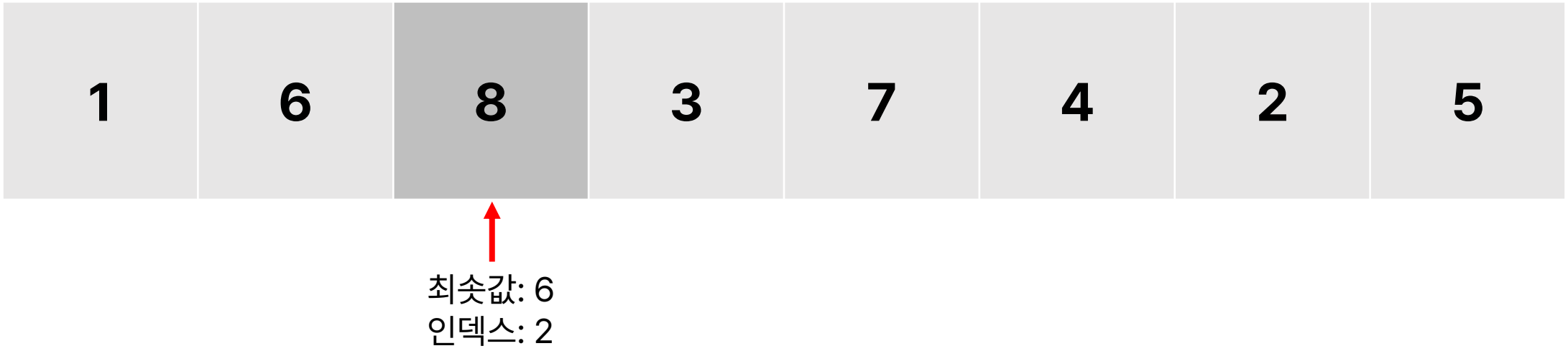
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

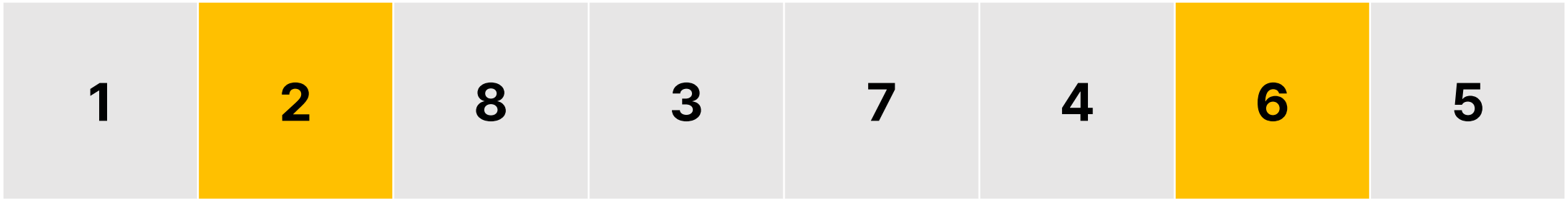


선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다

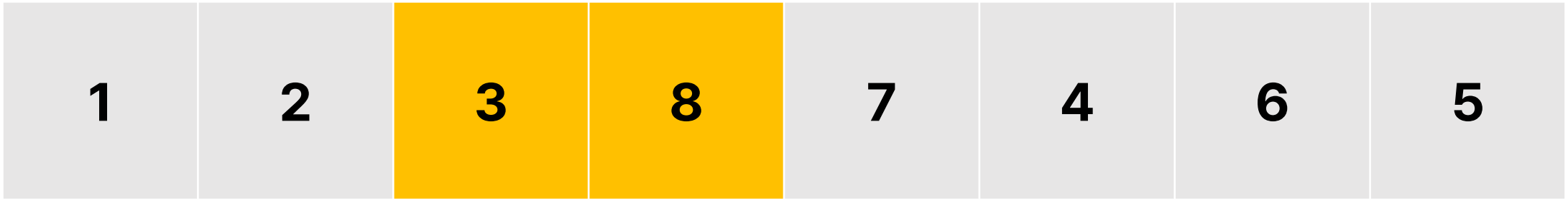
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



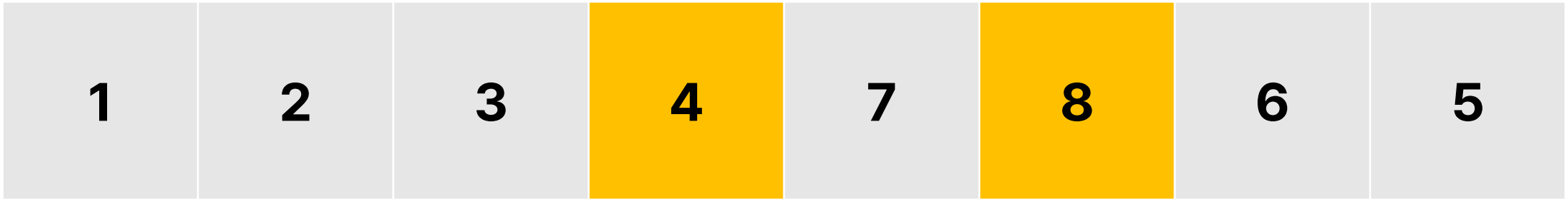
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



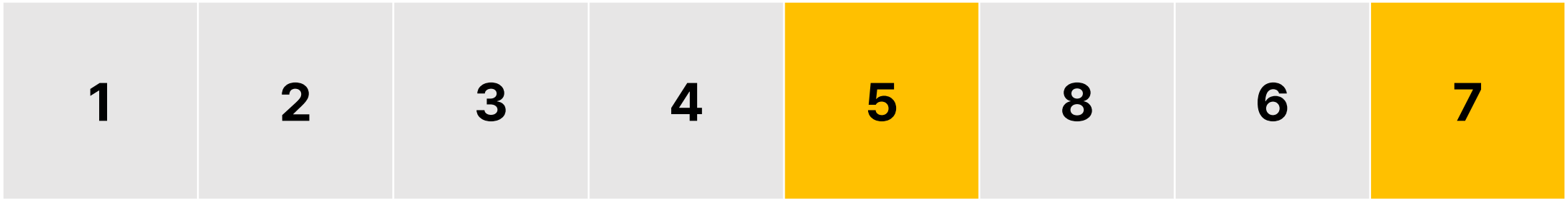
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



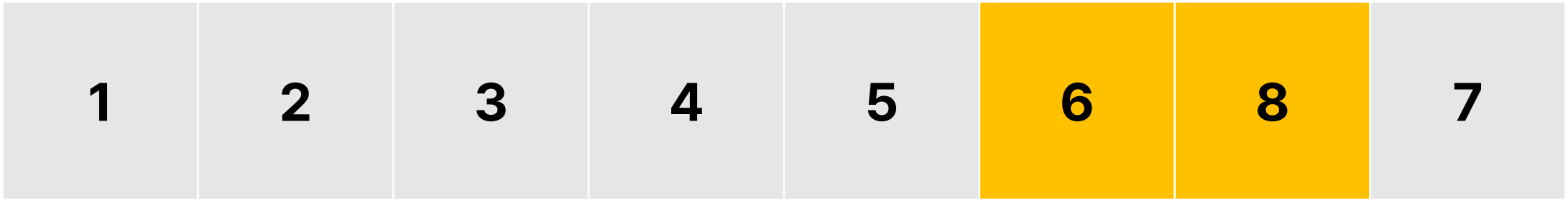
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



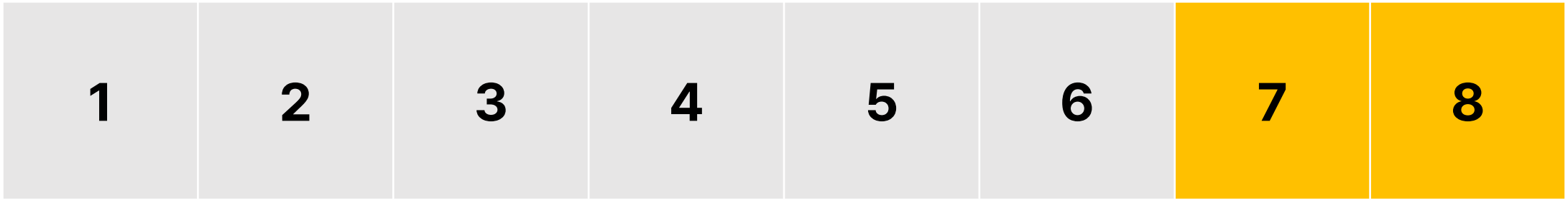
선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



선택 정렬 (Selection Sort)

- 주어진 원소 중 최솟값을 찾는다
- 해당 값을 맨 앞에 위치한 값과 교체한다
- 비교를 시작한 다음 위치부터 위 행동을 반복한다



삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다

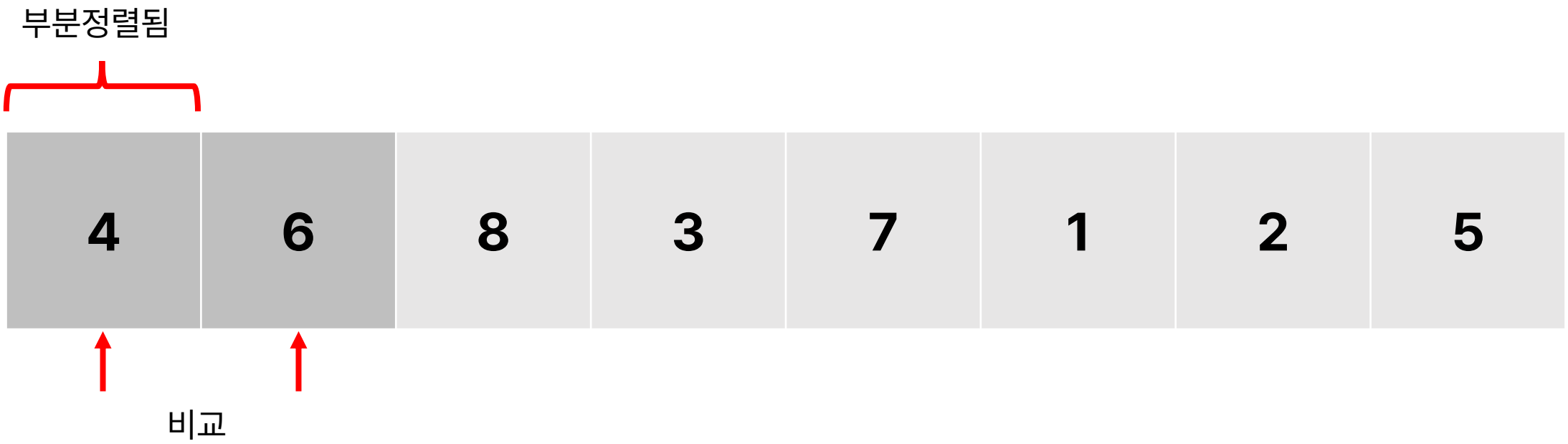
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다

4	6	8	3	7	1	2	5

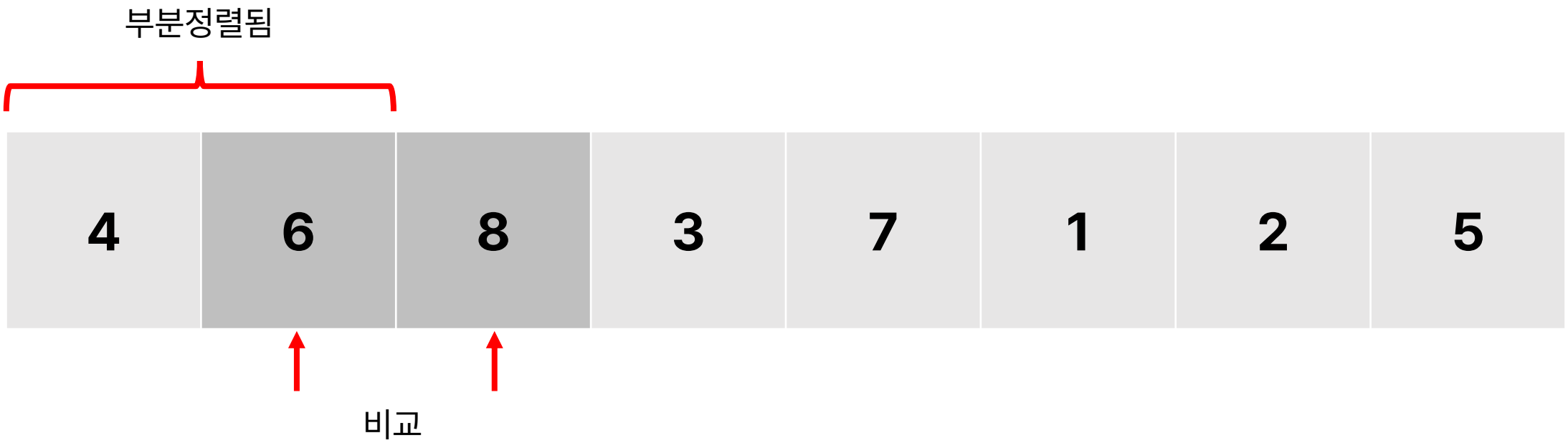
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



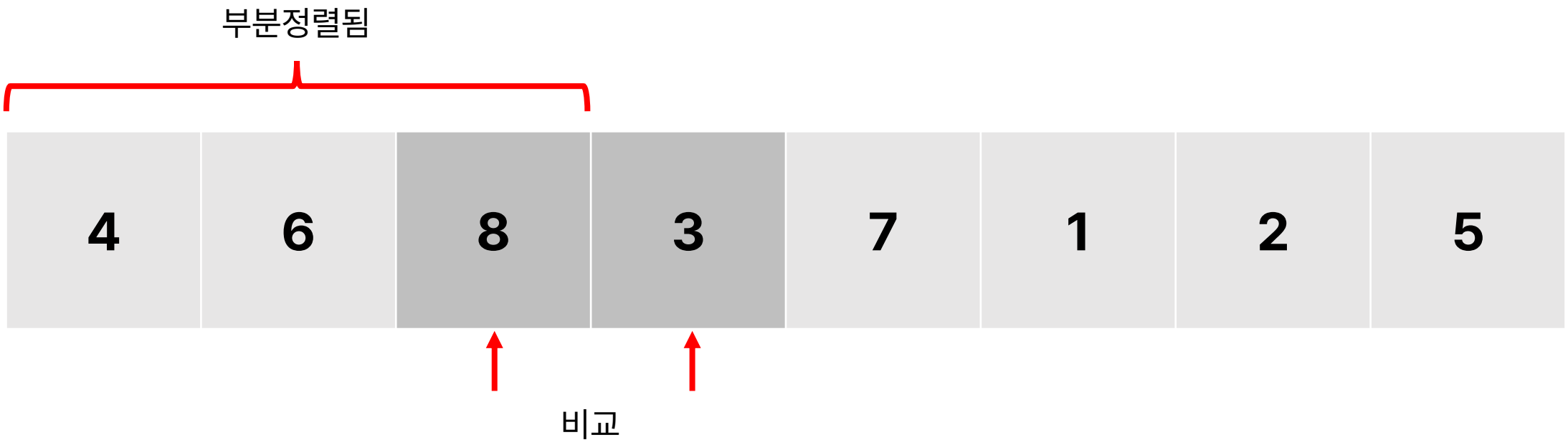
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



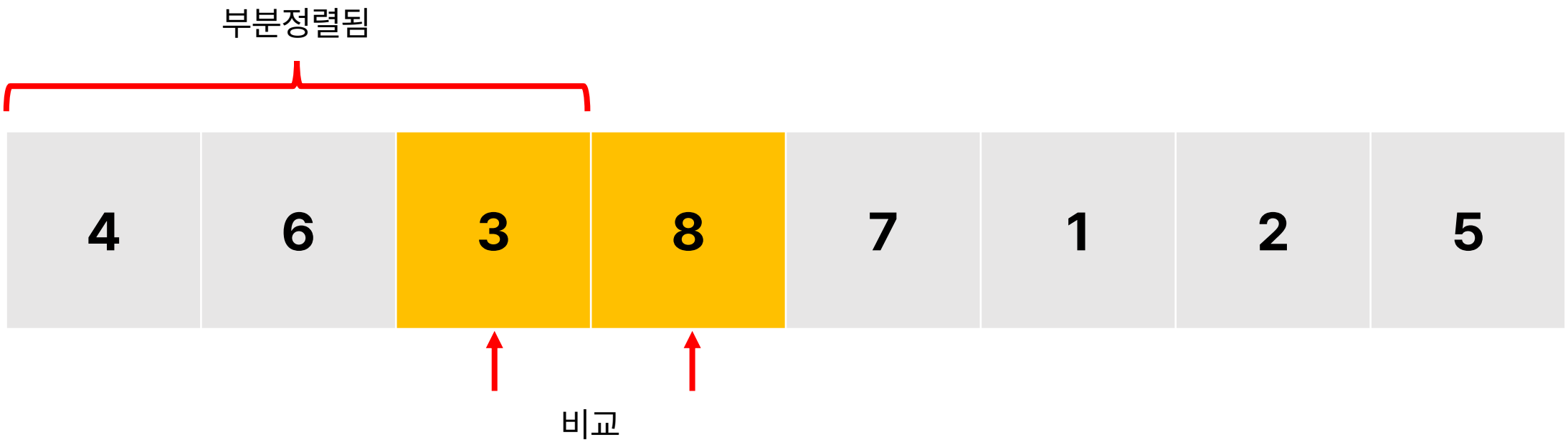
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



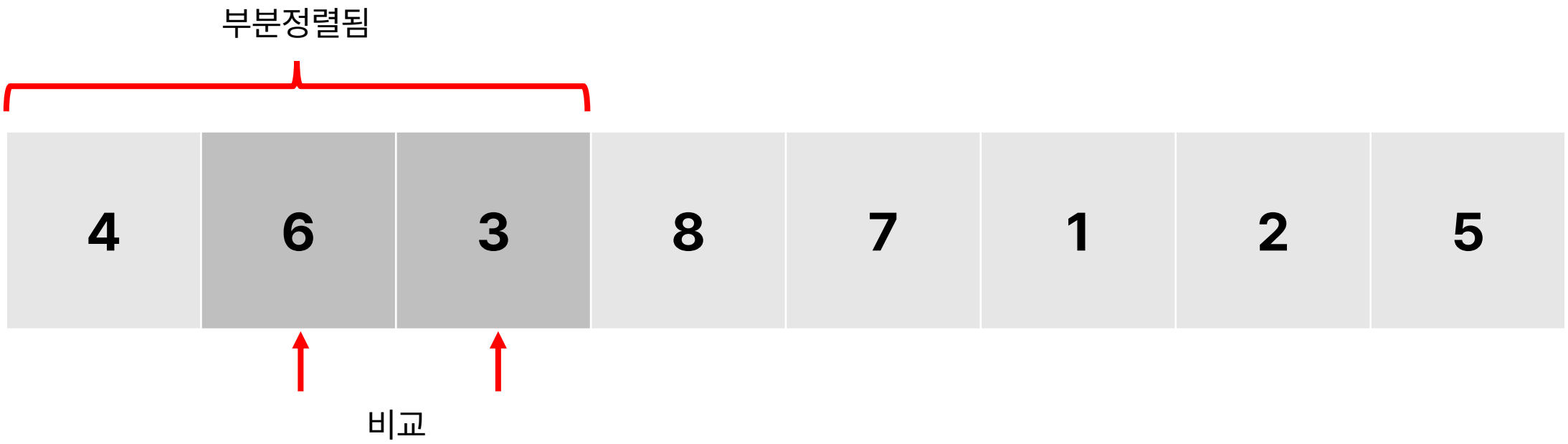
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



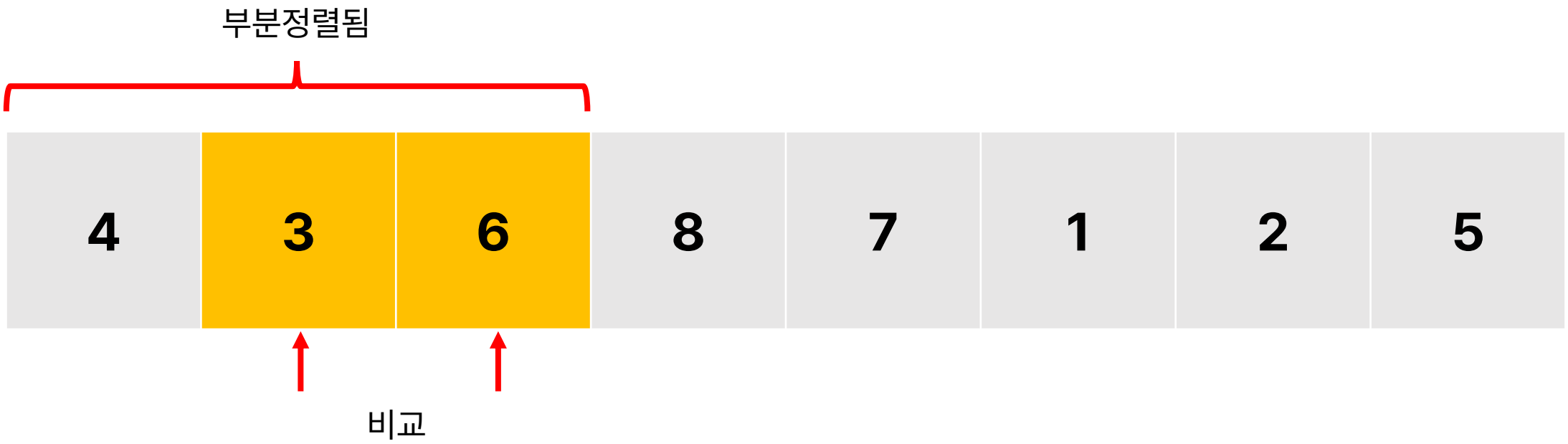
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



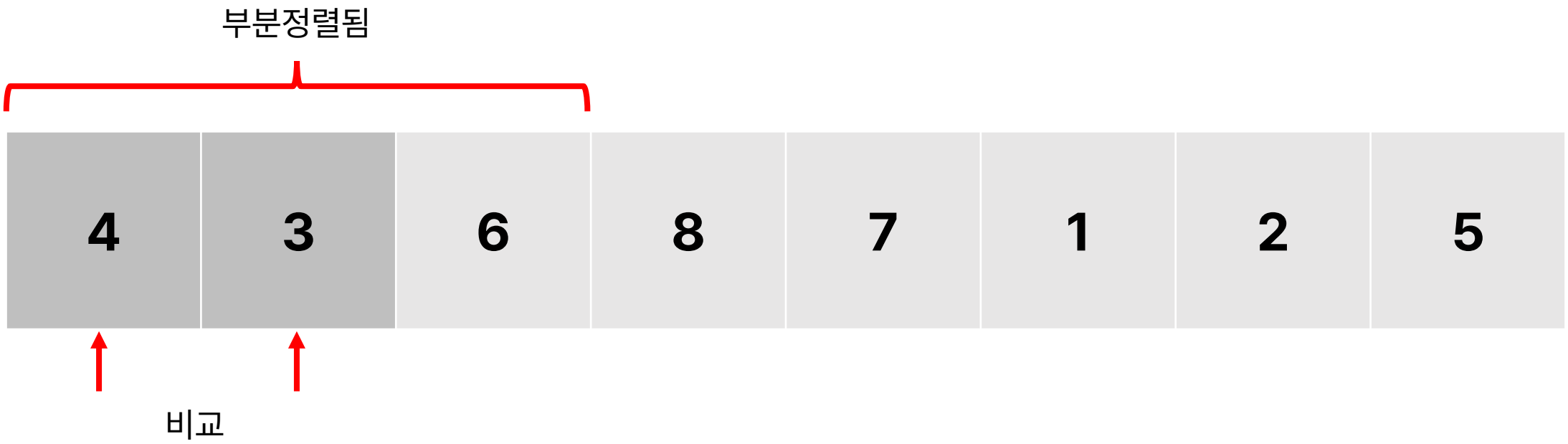
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



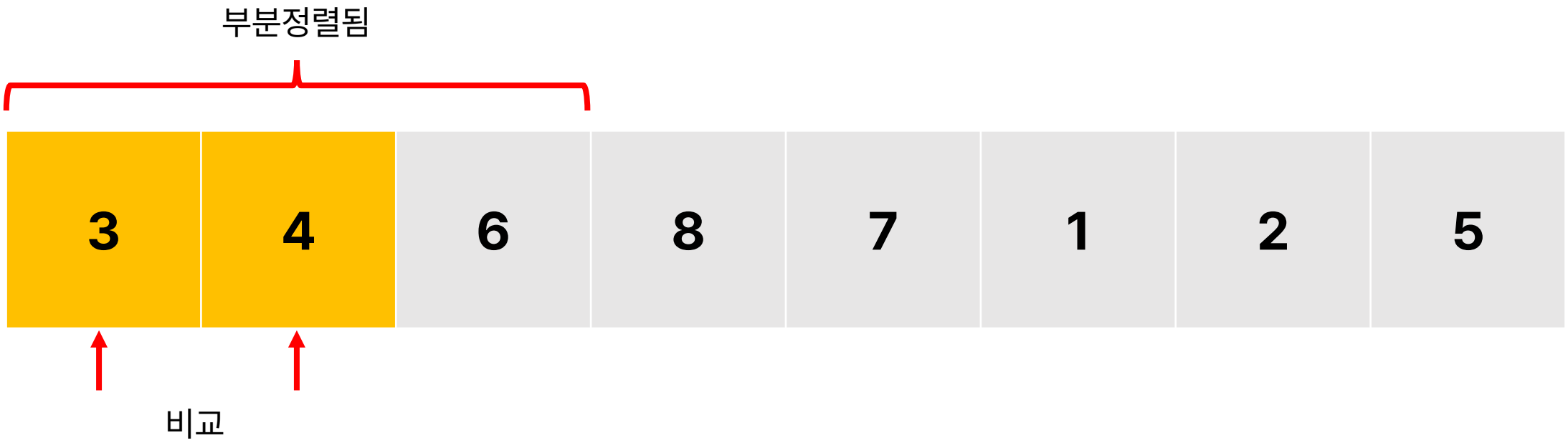
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



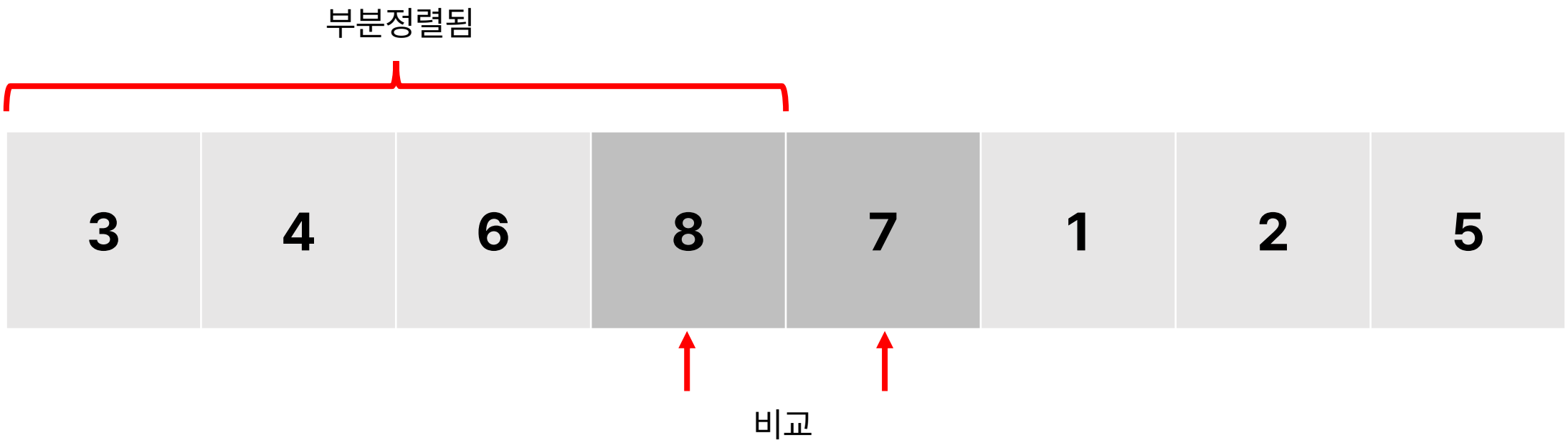
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



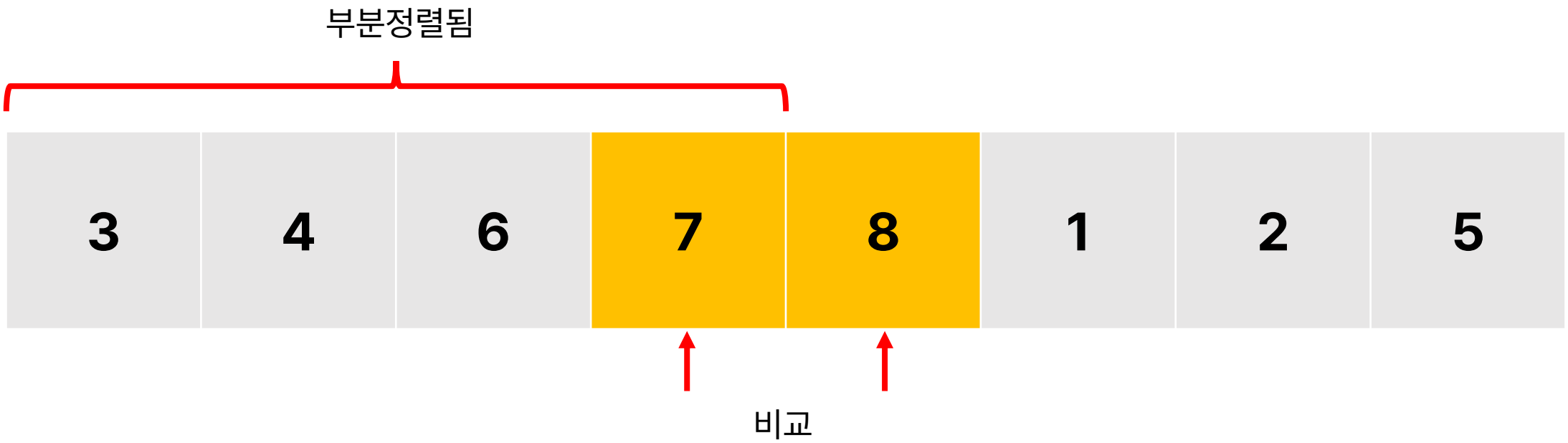
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



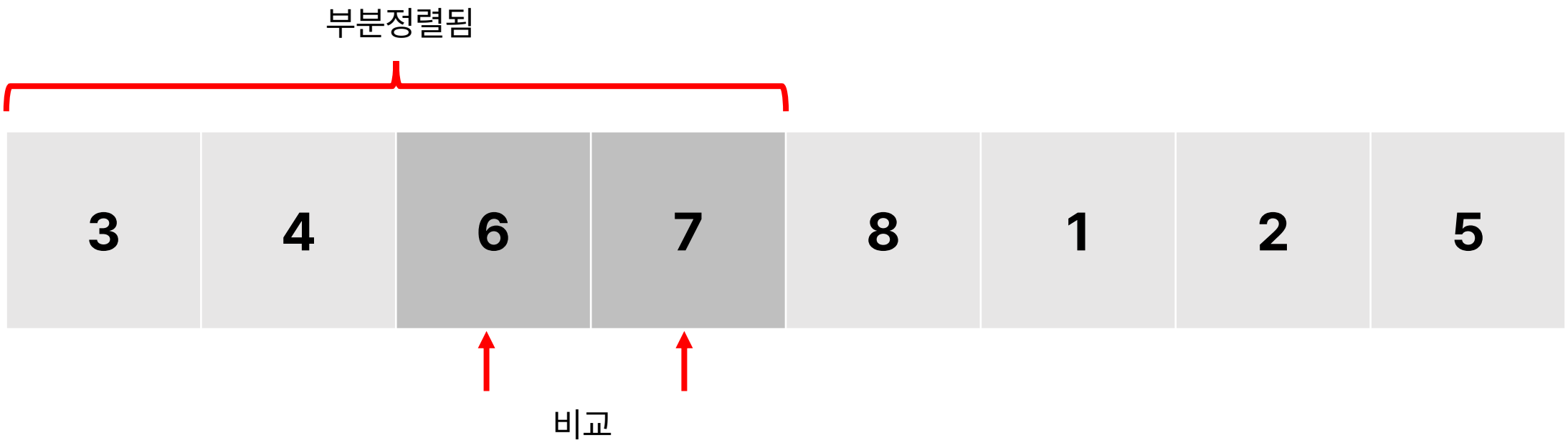
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



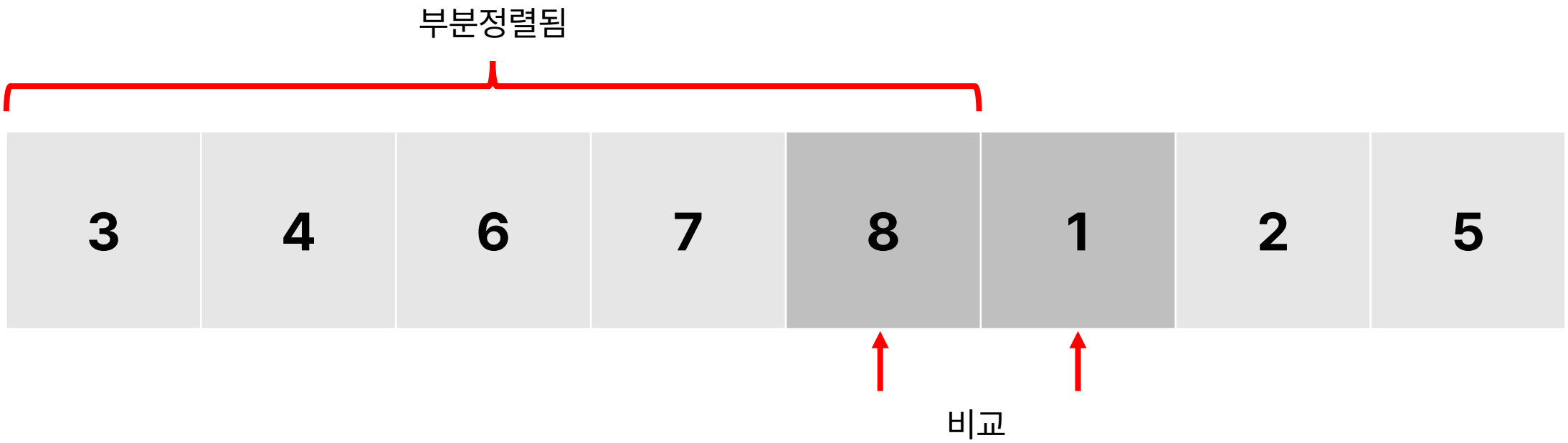
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



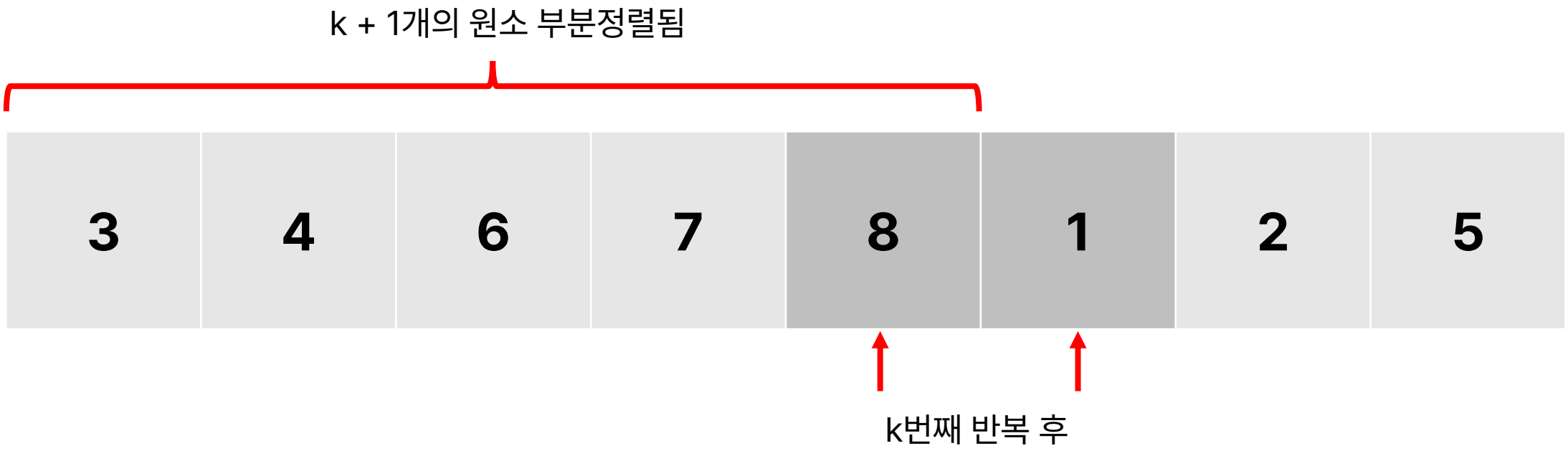
삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



삽입 정렬 (Insertion Sort)

- 현재 위치의 원소를 앞에서부터 차례대로 정렬된 배열 부분과 비교해 삽입한다
- 다음 위치로 이동한다



버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다

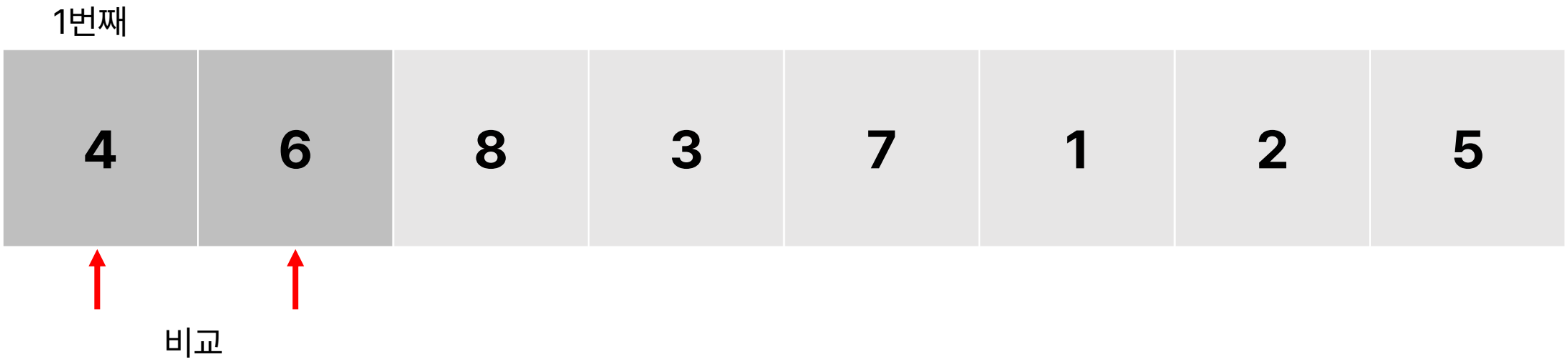
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다

4	6	8	3	7	1	2	5

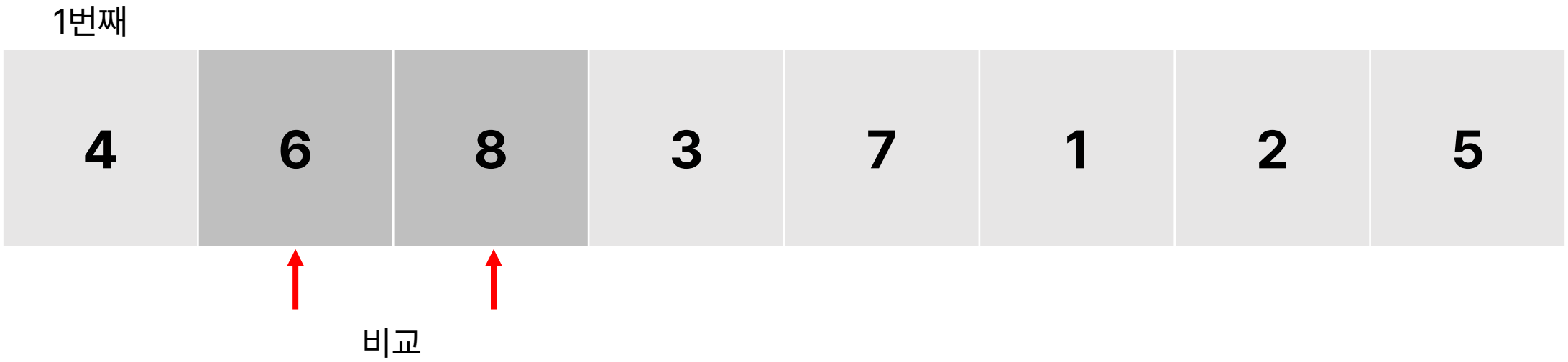
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



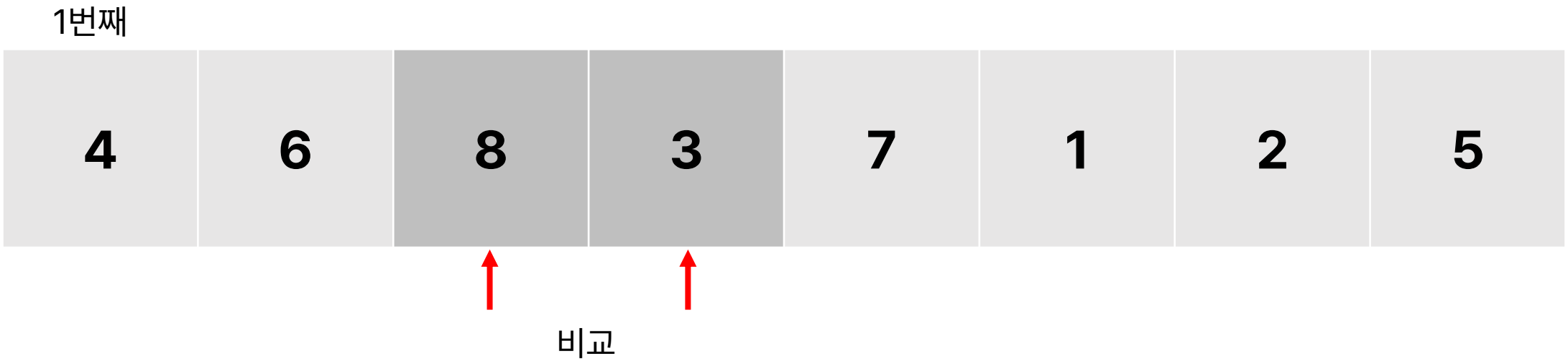
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



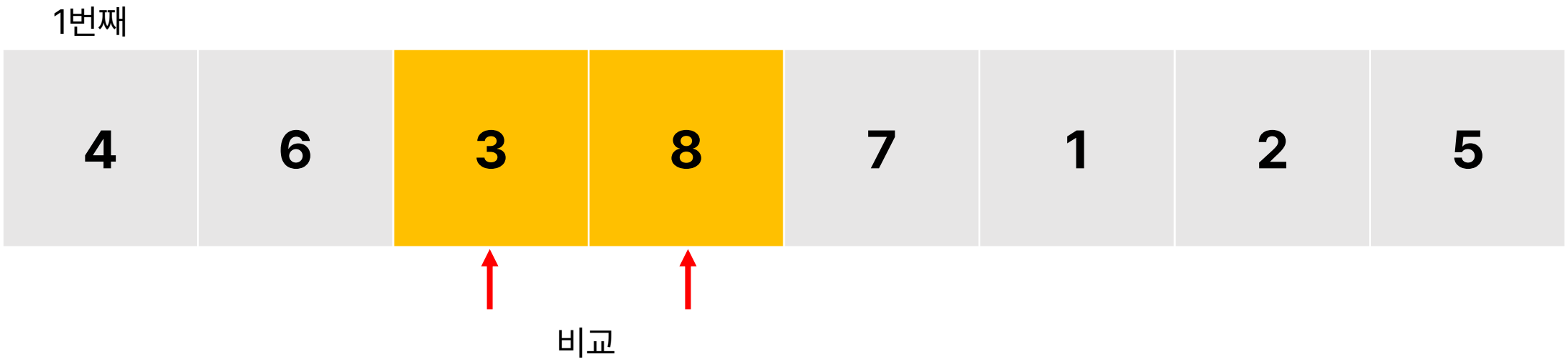
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



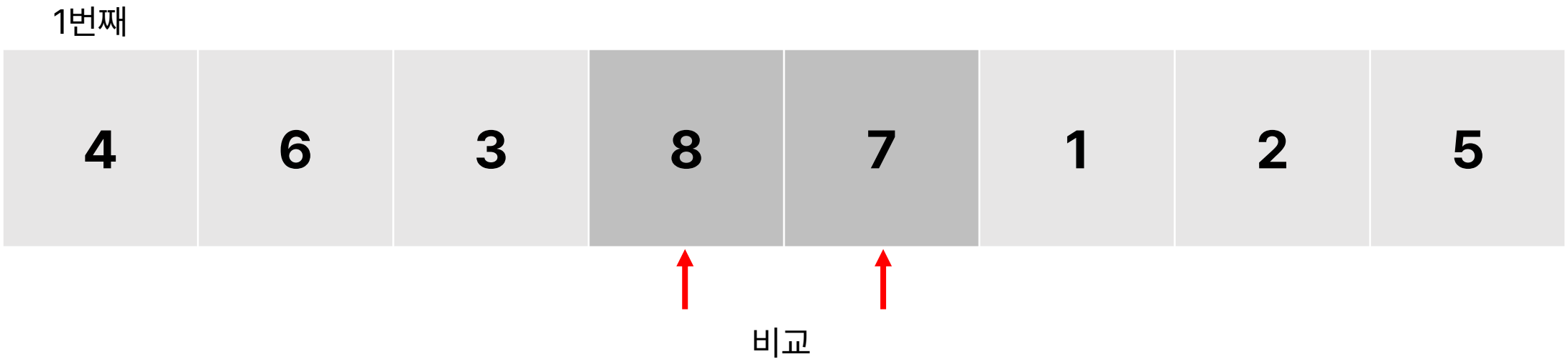
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



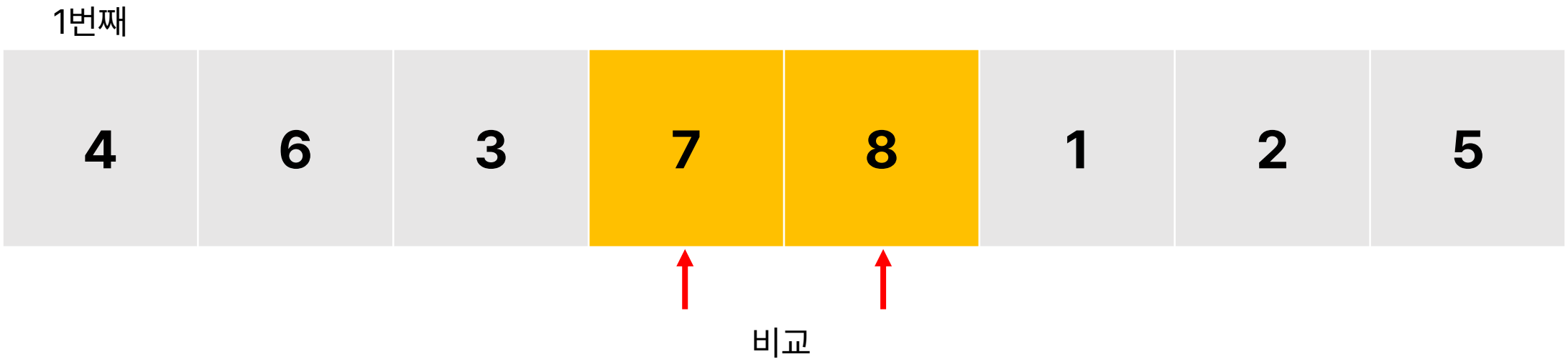
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



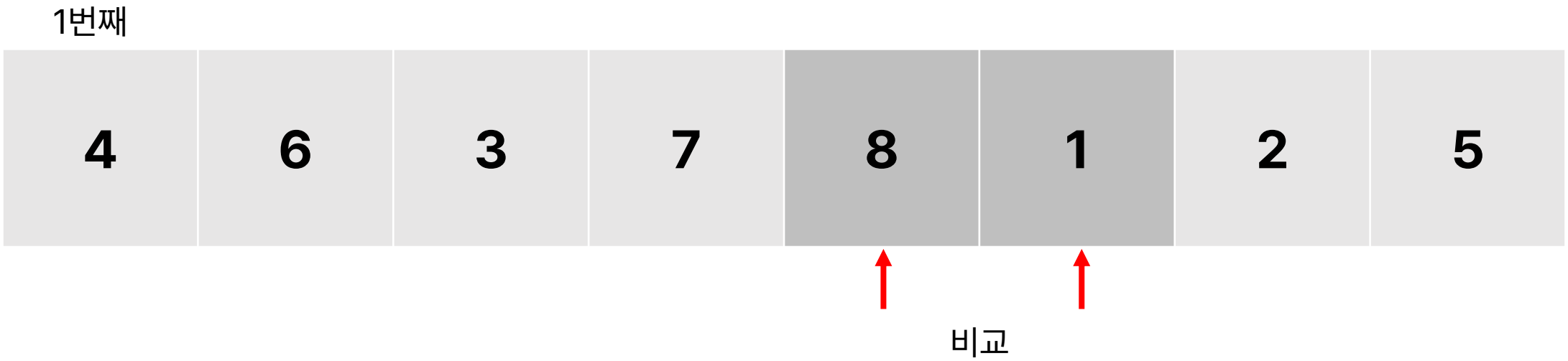
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



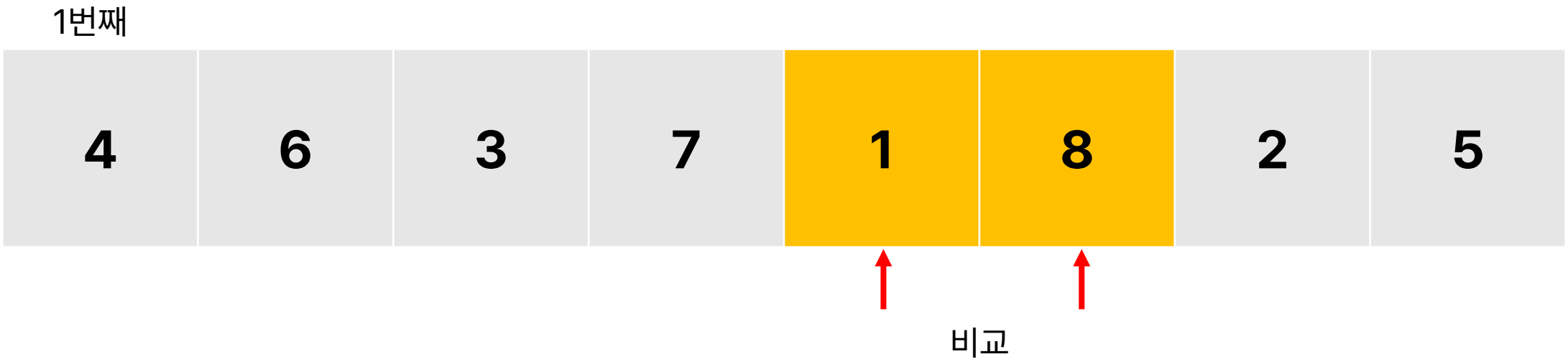
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



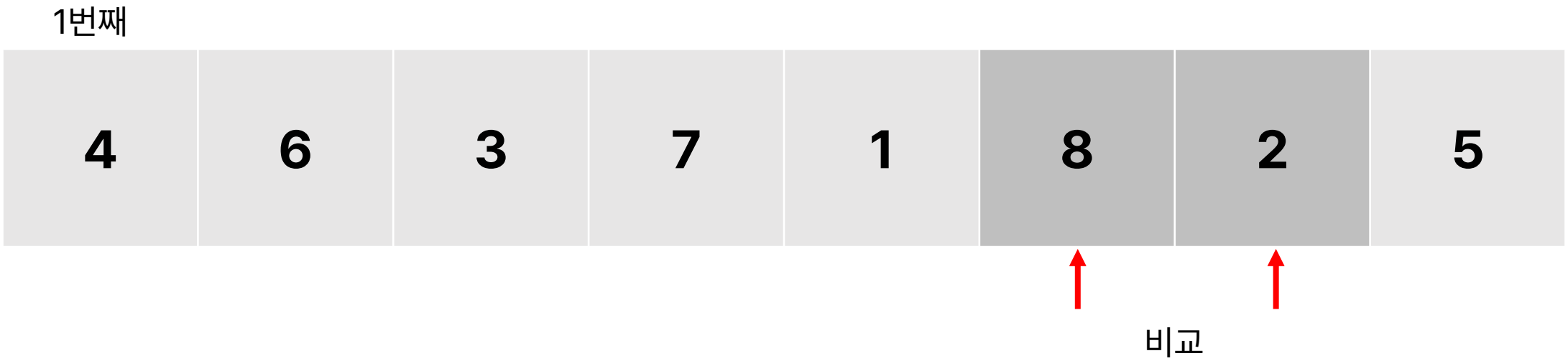
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



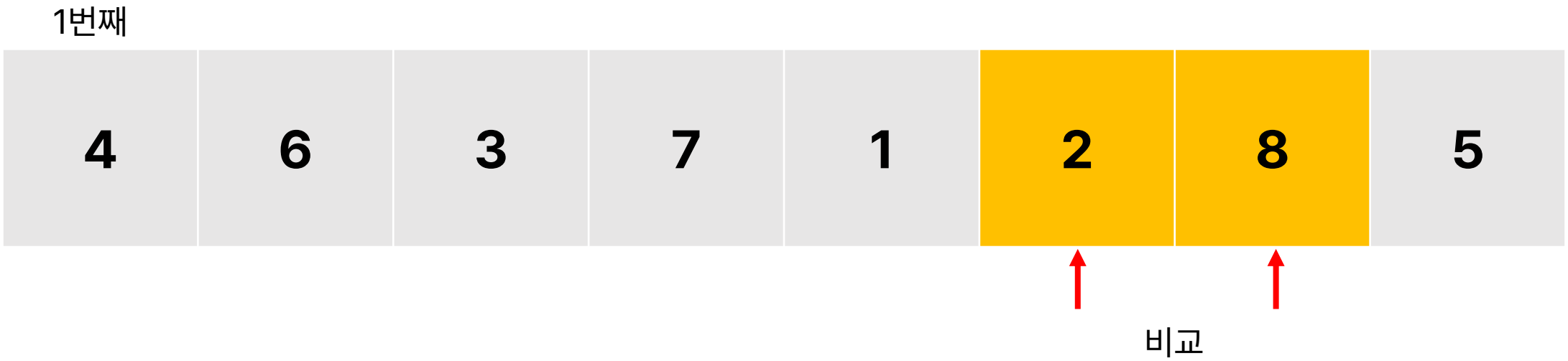
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



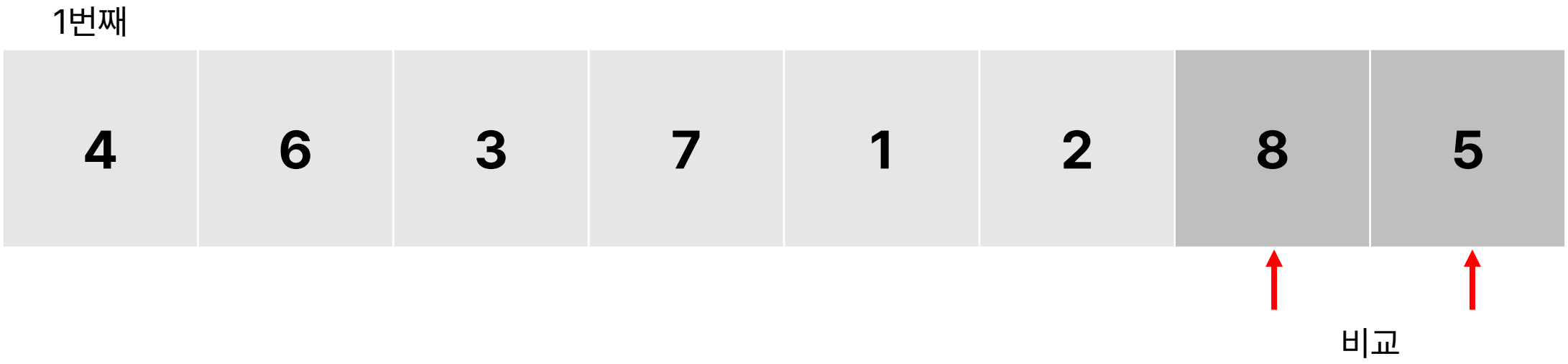
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



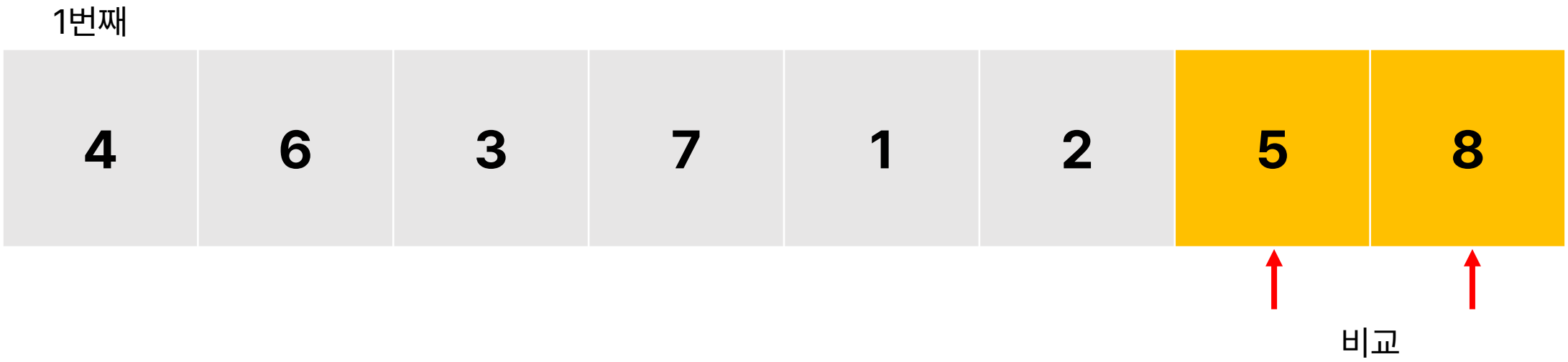
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



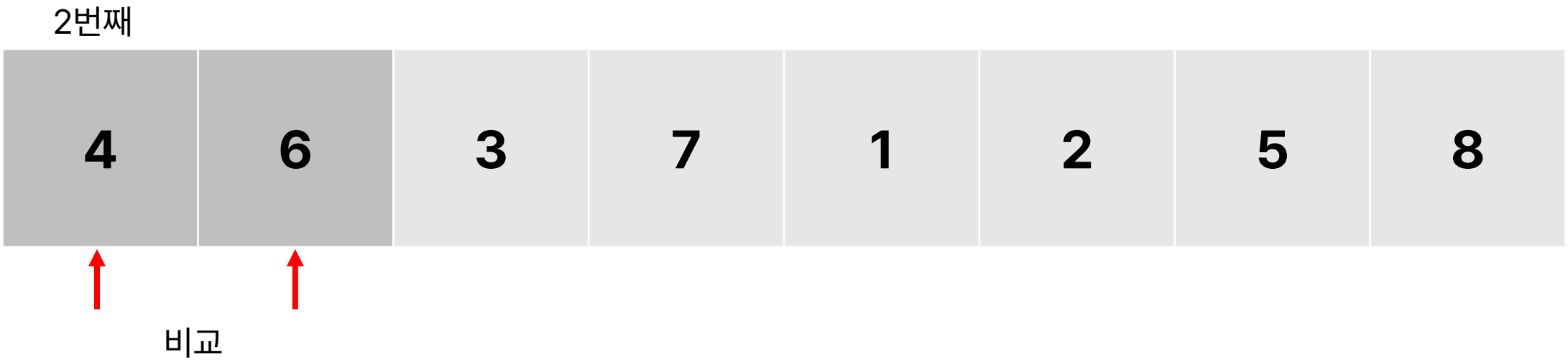
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



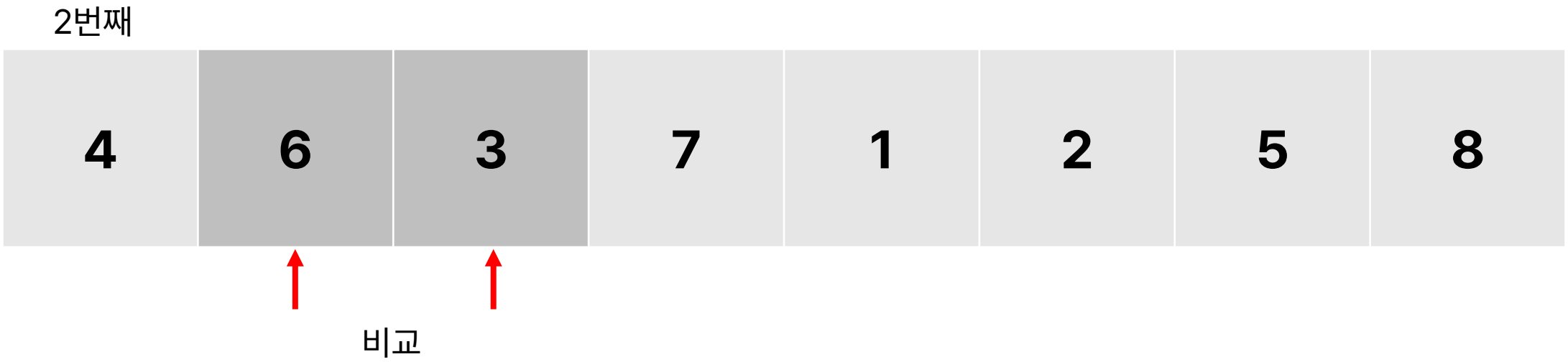
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



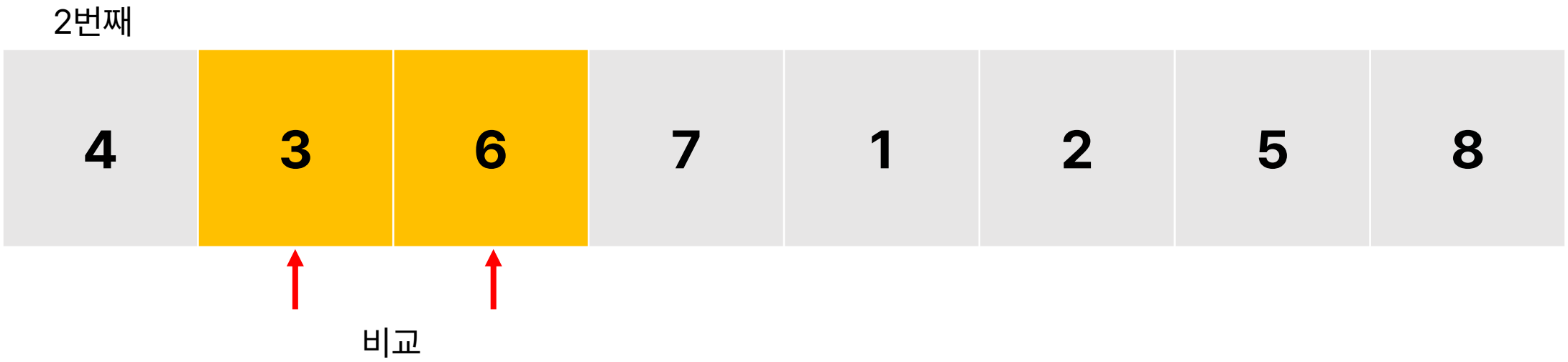
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



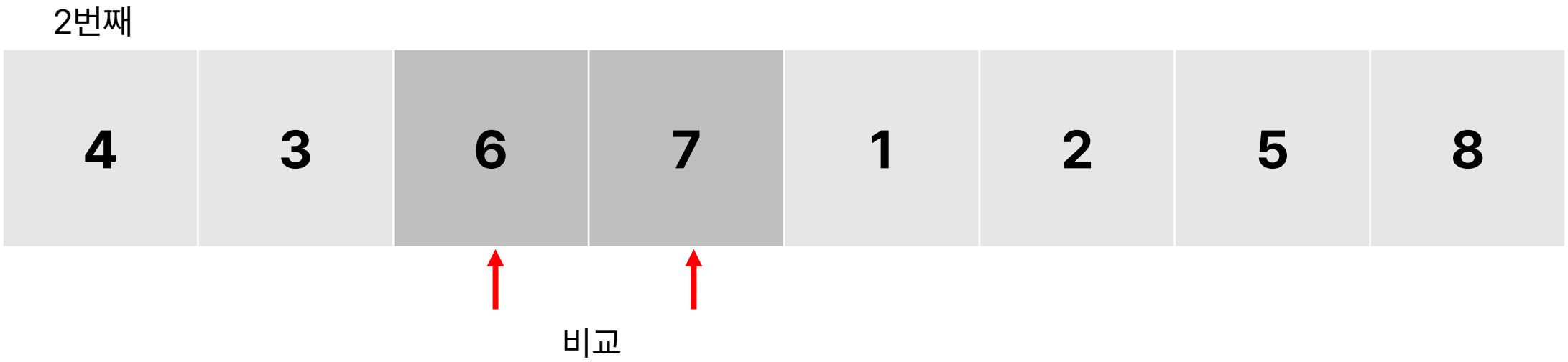
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



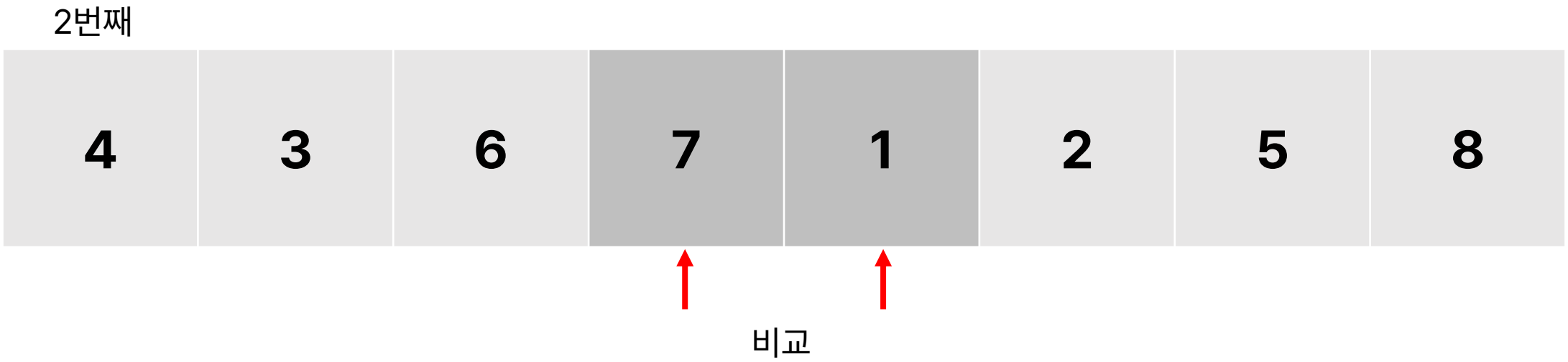
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



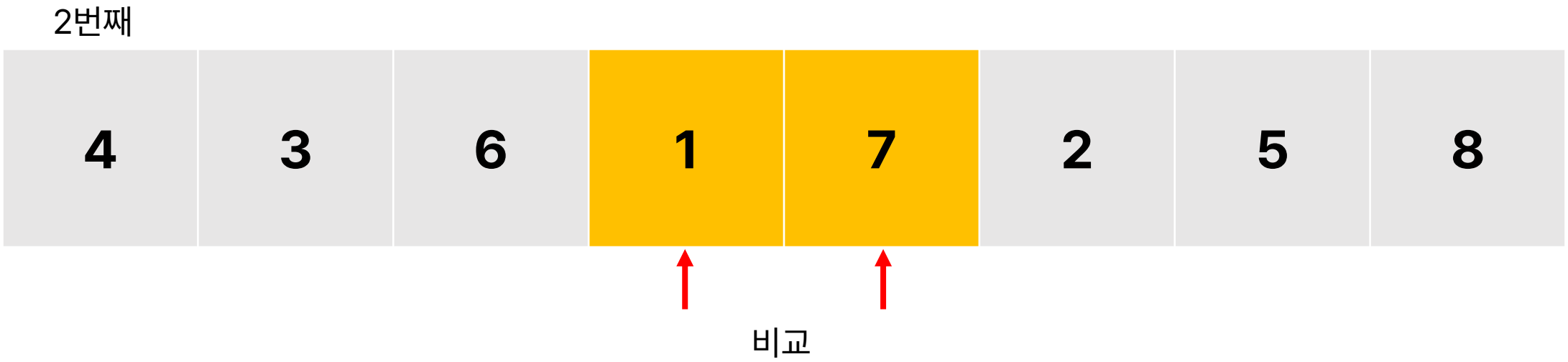
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



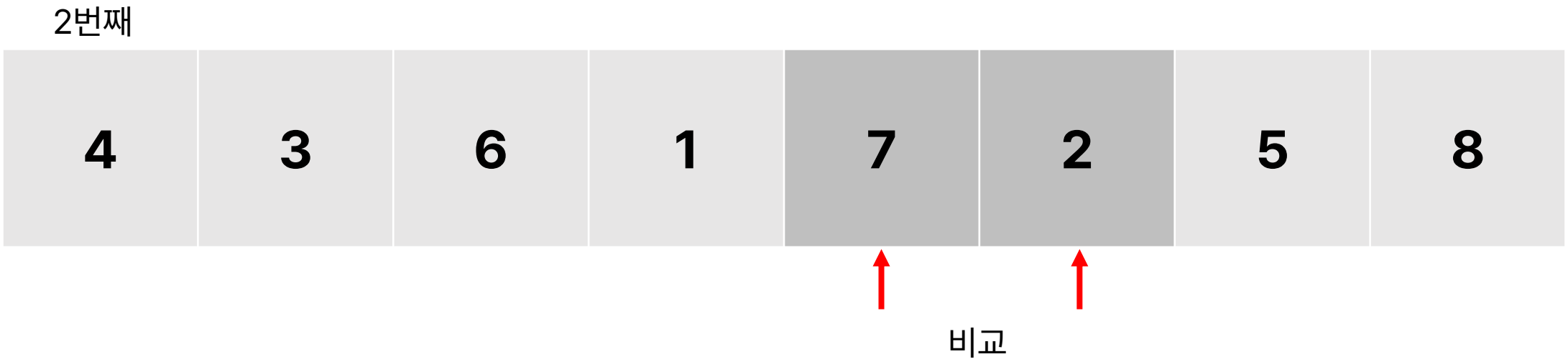
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



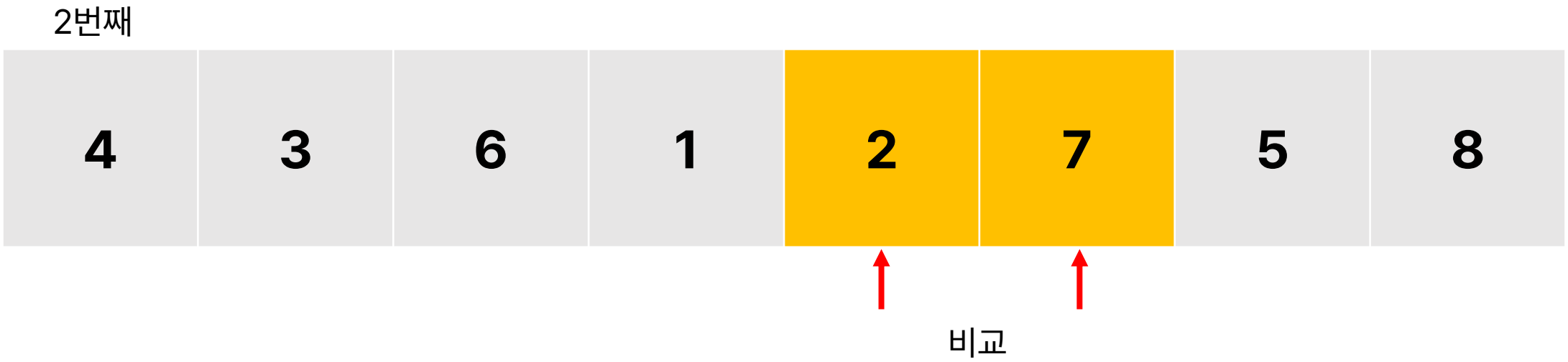
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



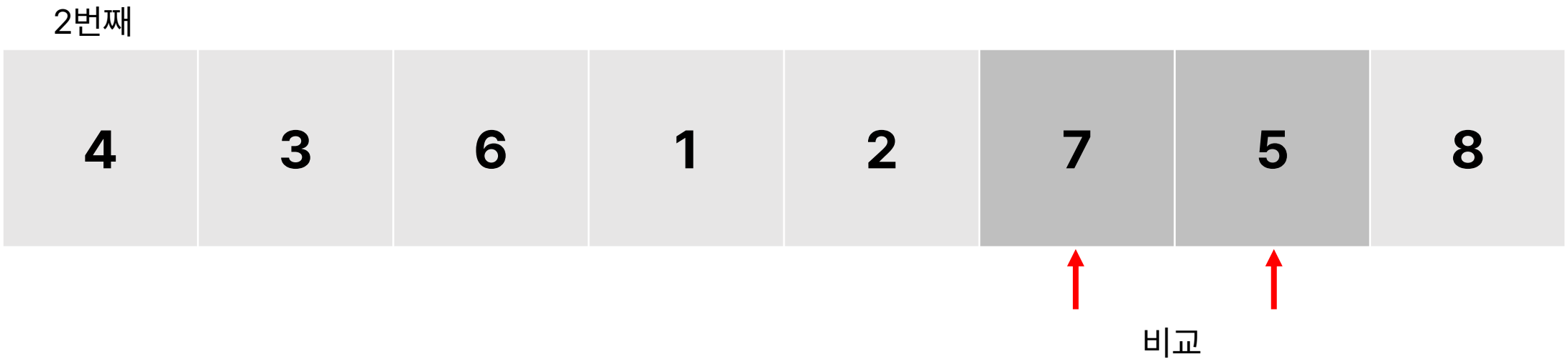
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



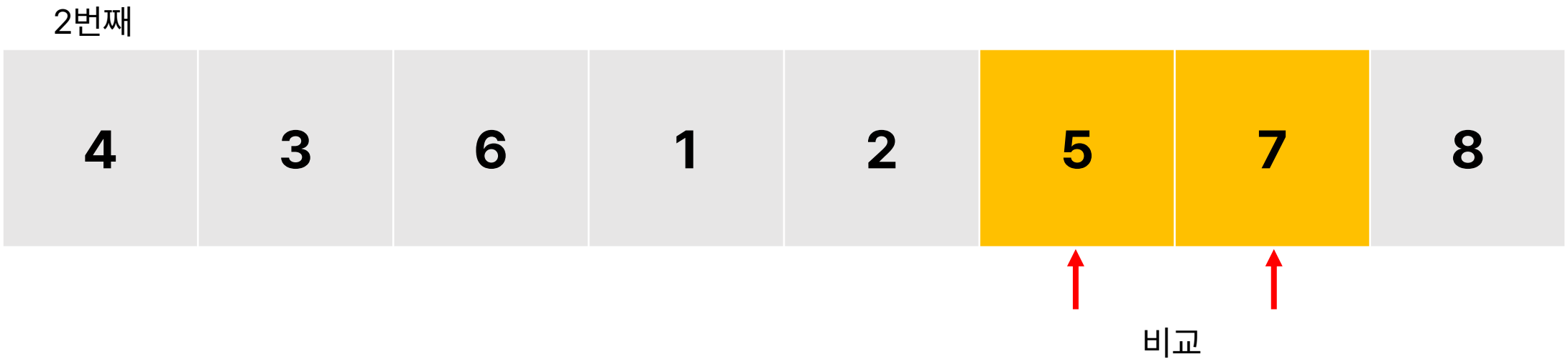
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



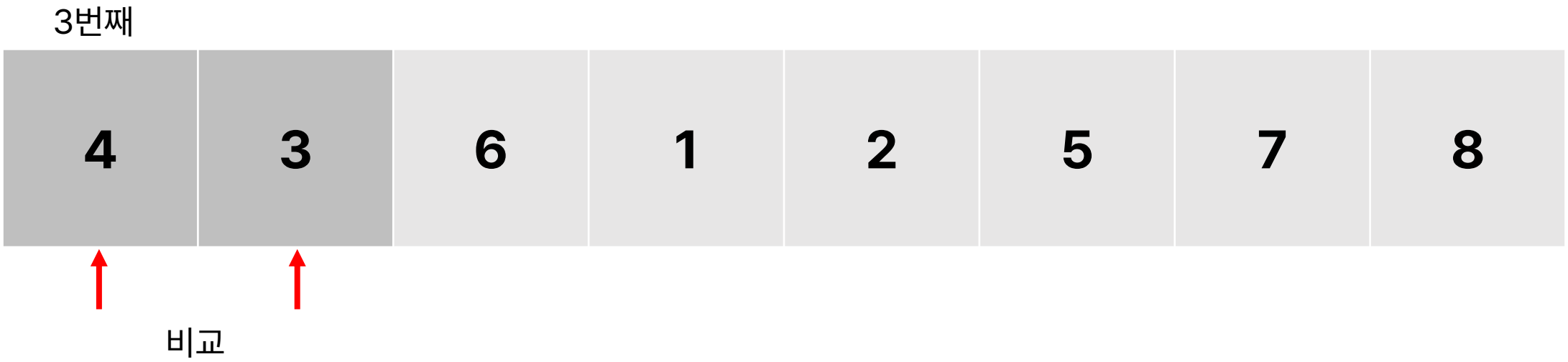
버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



버블 정렬 (Bubble Sort)

- 인접한 두 원소를 선택한다
  - 정렬되어 있다면 그대로 놔두고, 아니라면 바꾼다
  - 배열의 처음부터 끝까지 반복한다
- 배열에 변화가 없을 때까지 반복한다



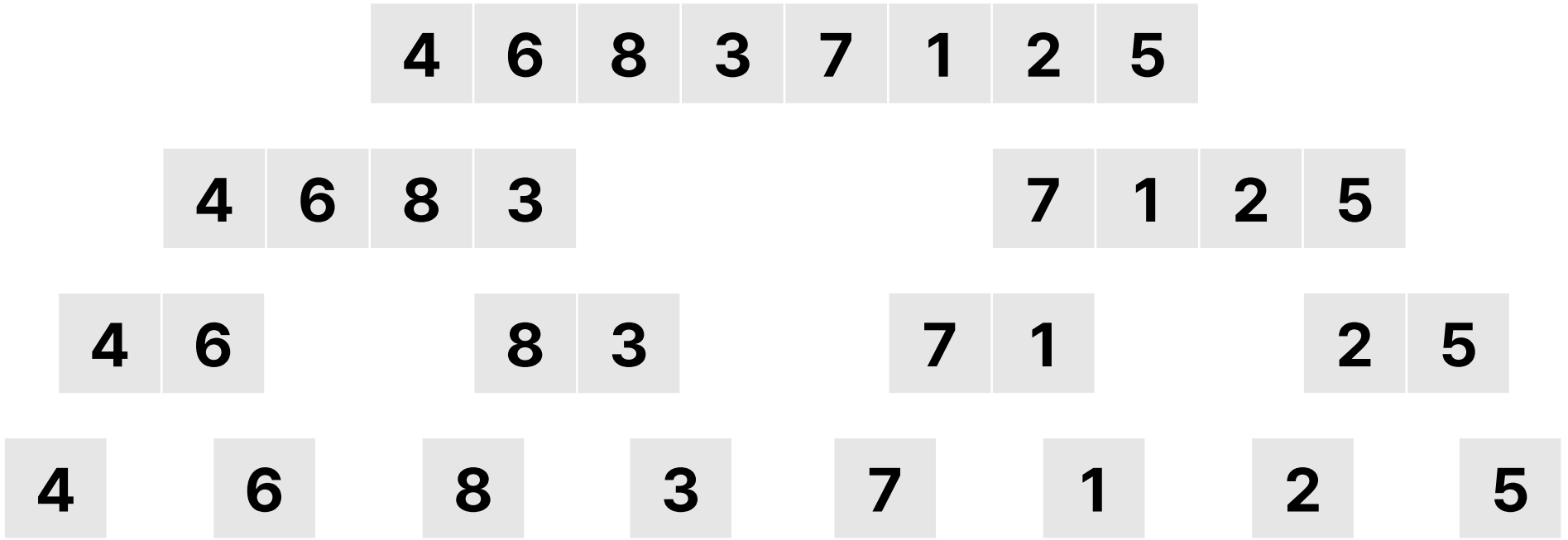
수 정렬하기 BOJ 2750

- N개의 수가 주어졌을 때, 이를 오름차순으로 정렬하는 프로그램을 작성하시오
- $N(1 \leq N \leq 1\,000)$, 주어지는 수는 절댓값이 $1\,000$보다 작거나 같은 정수

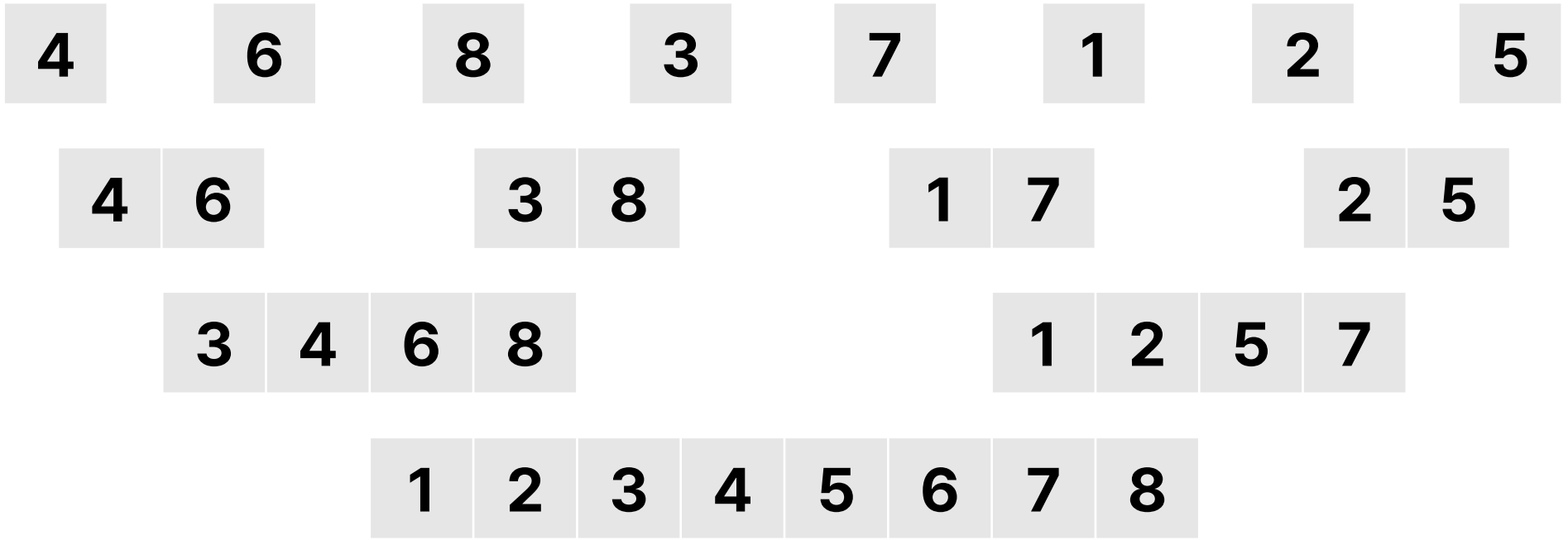
병합 정렬 (Merge Sort)

- 배열의 길이가 1 이하이면 정렬된 상태이다
- 분할 (divide)
  - 정렬되지 않은 배열을 절반으로 잘라 두 부분 배열로 분할한다
- 정복 (conquer)
  - 각 부분 배열을 재귀적으로 정렬을 진행한다
- 결합 (combine)
  - 두 부분 배열을 하나의 정렬된 배열로 병합한다 (임시 배열에 저장)
- 복사 (copy)
  - 임시 배열에 저장된 결과를 원래 배열에 복사한다

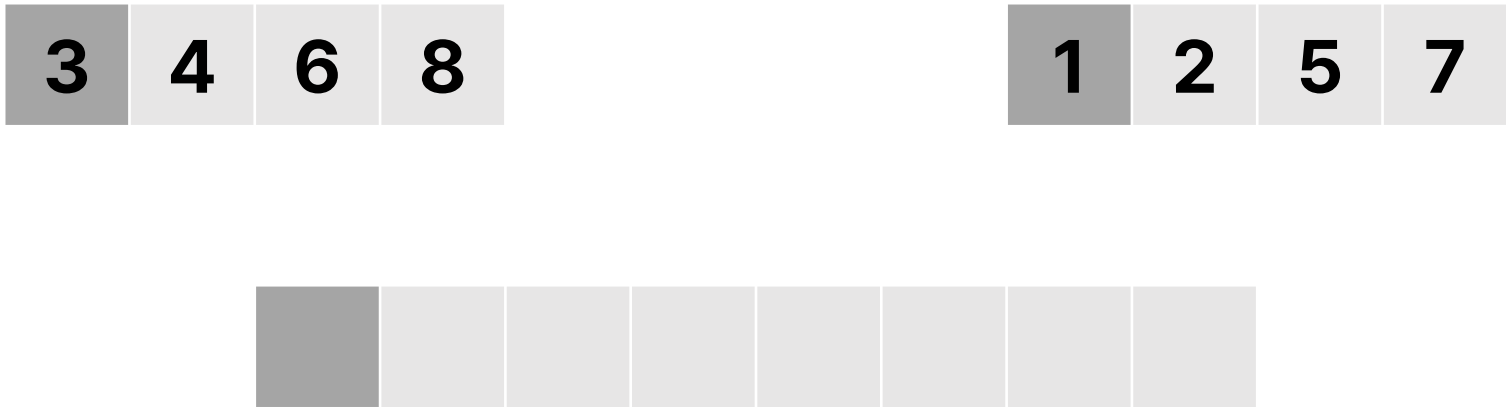
병합 정렬 (Merge Sort) - 분할



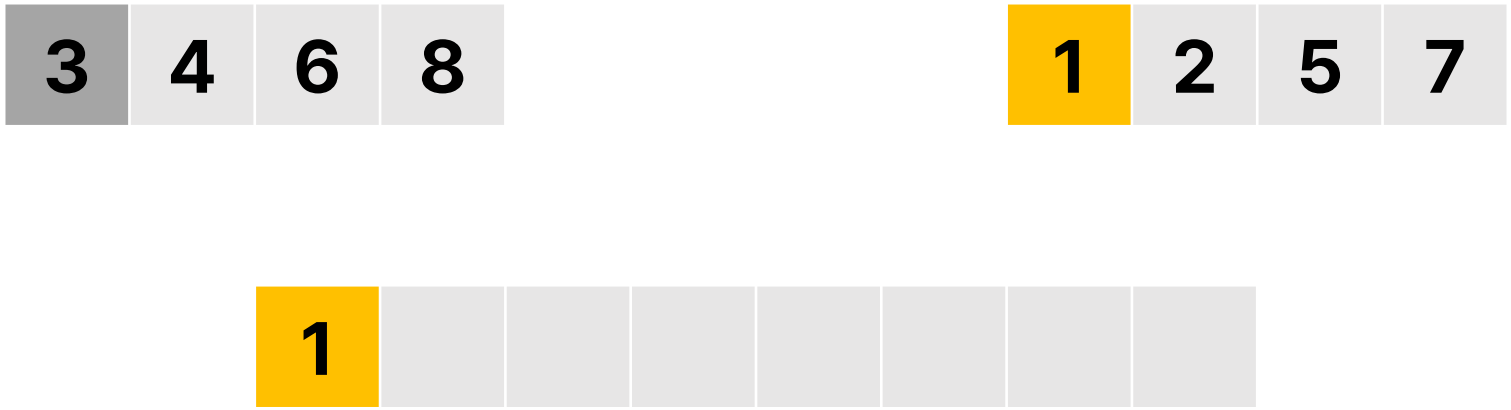
병합 정렬 (Merge Sort) - 정복 / 결합



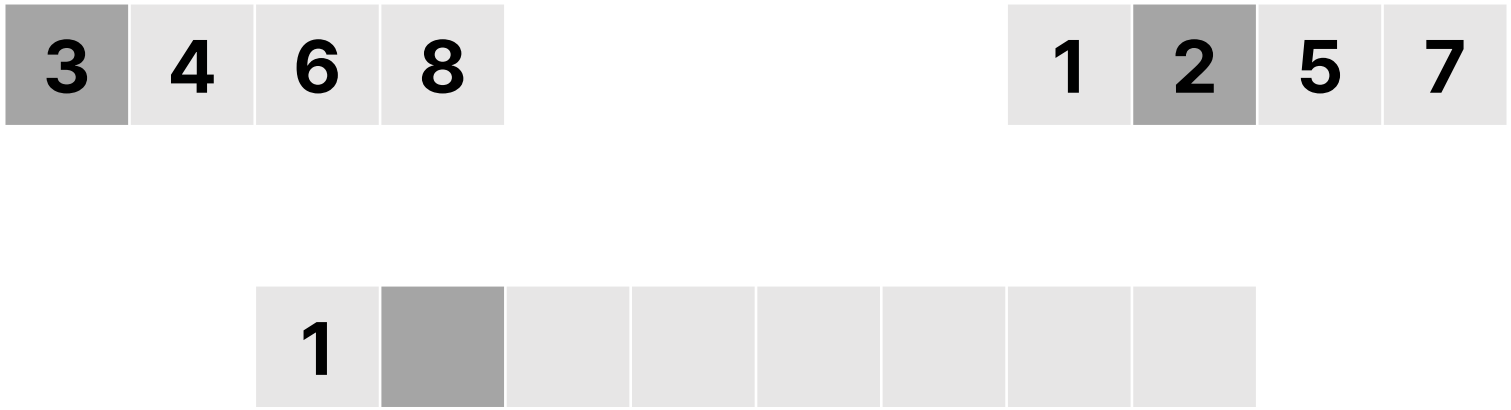
병합 정렬 (Merge Sort) - 결합



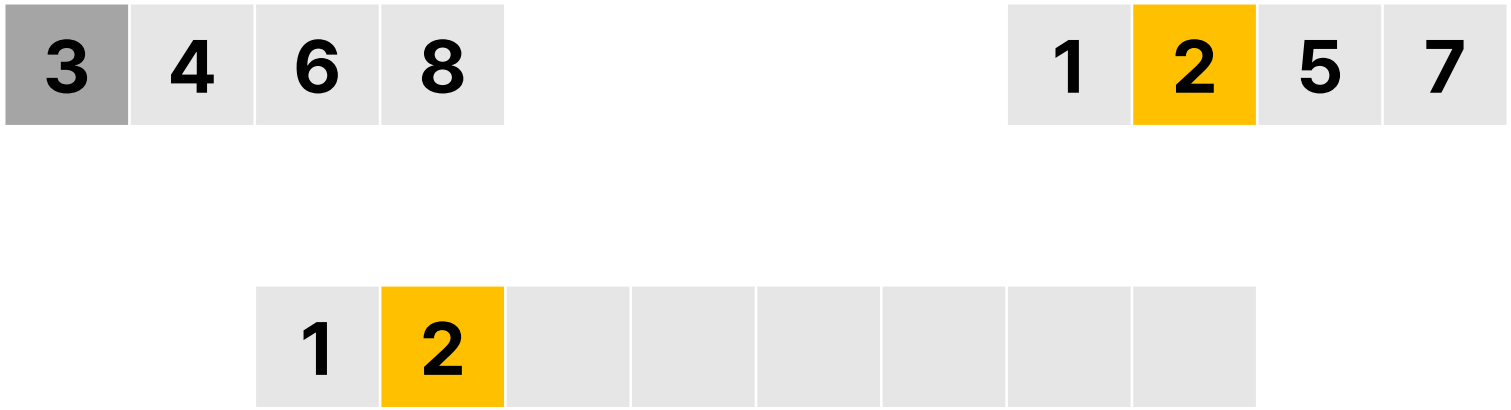
병합 정렬 (Merge Sort) - 결합



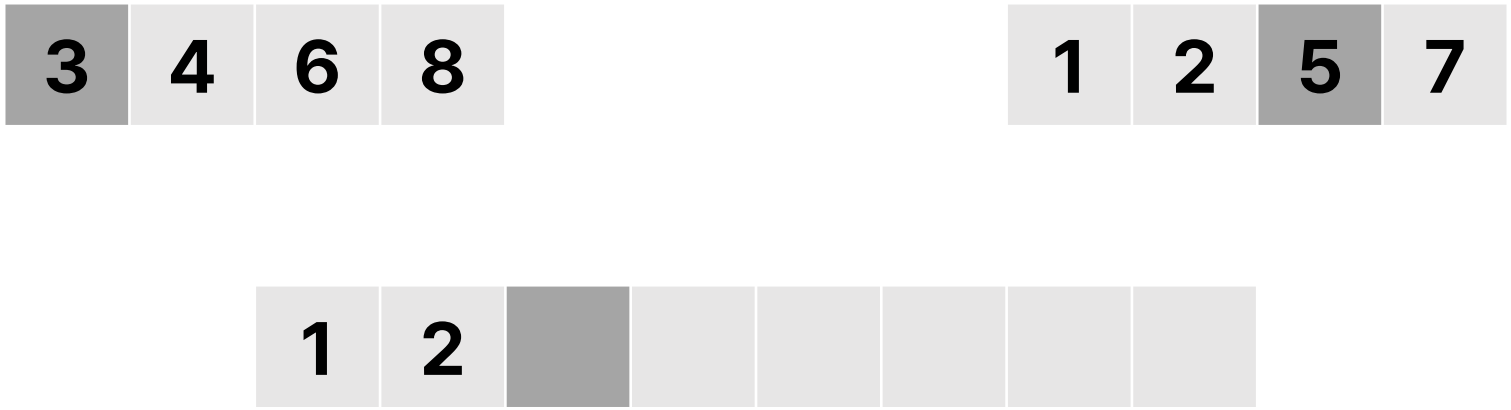
병합 정렬 (Merge Sort) - 결합



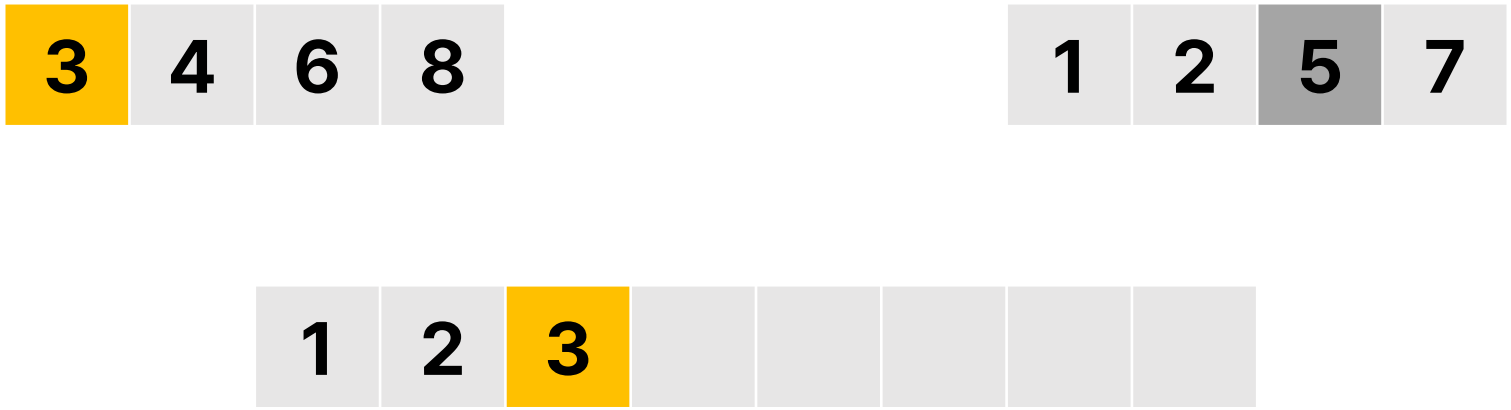
병합 정렬 (Merge Sort) - 결합



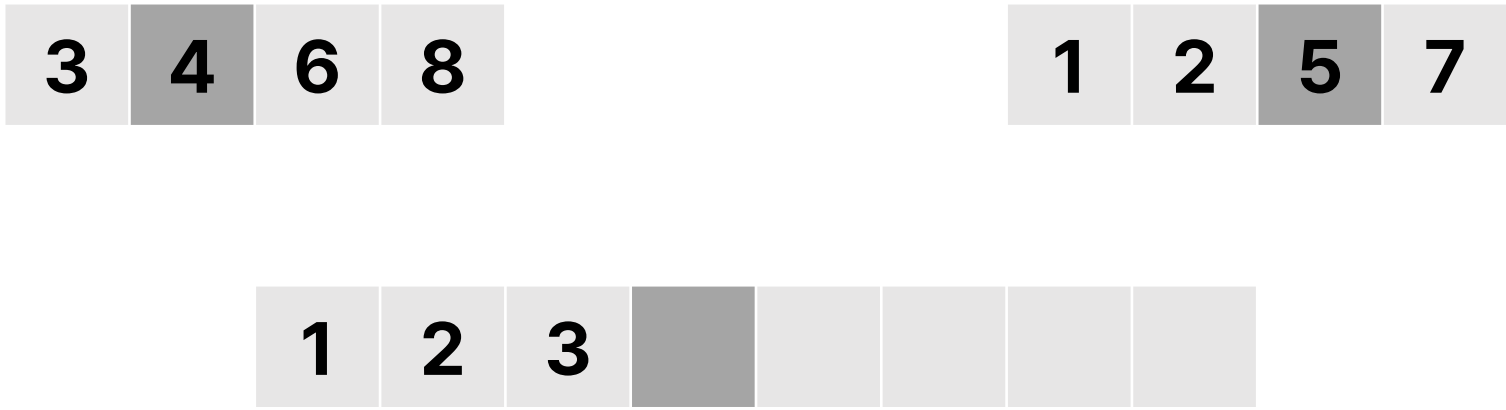
병합 정렬 (Merge Sort) - 결합



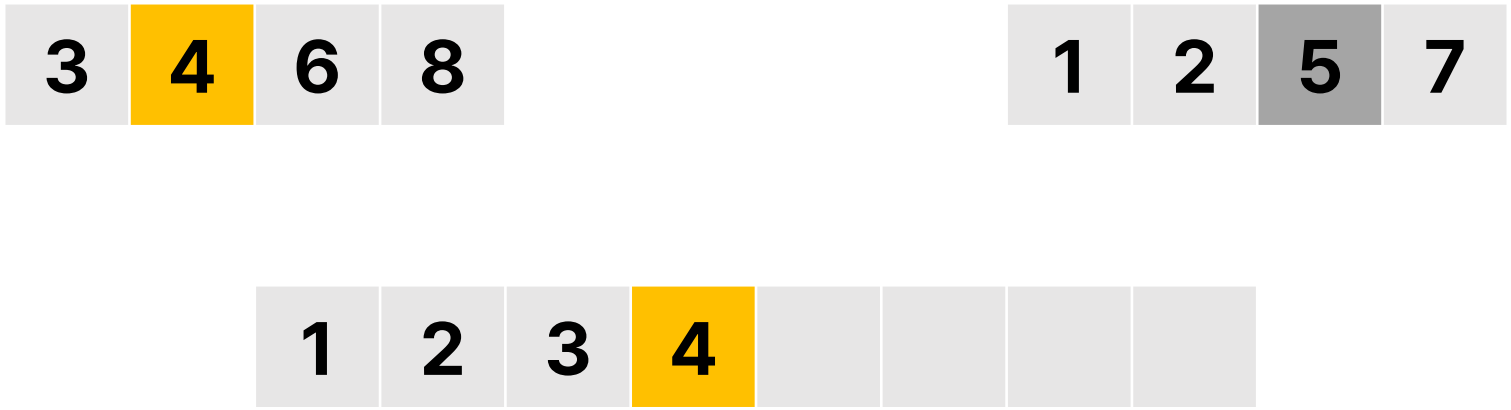
병합 정렬 (Merge Sort) - 결합



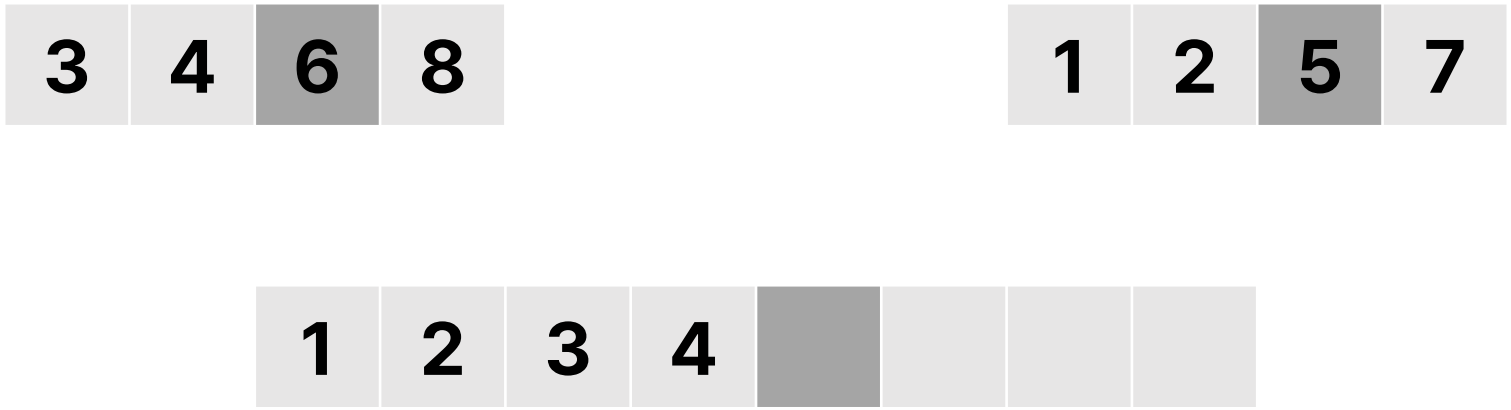
병합 정렬 (Merge Sort) - 결합



병합 정렬 (Merge Sort) - 결합

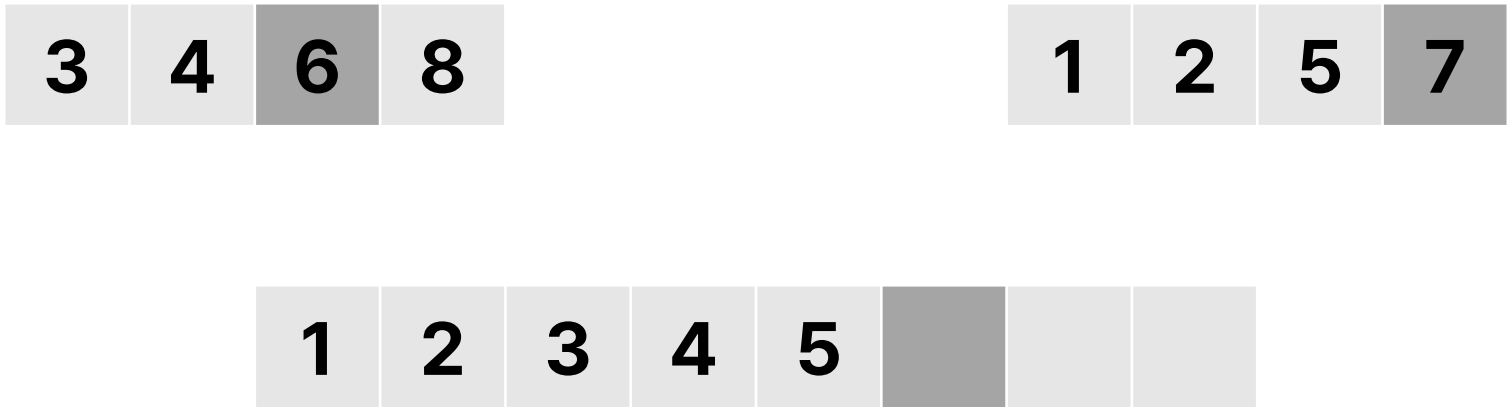


병합 정렬 (Merge Sort) - 결합



병합 정렬 (Merge Sort) - 결합

병합 정렬 (Merge Sort) - 결합



병합 정렬 (Merge Sort) - 결합

병합 정렬 (Merge Sort) - 결합

병합 정렬 (Merge Sort) - 결합

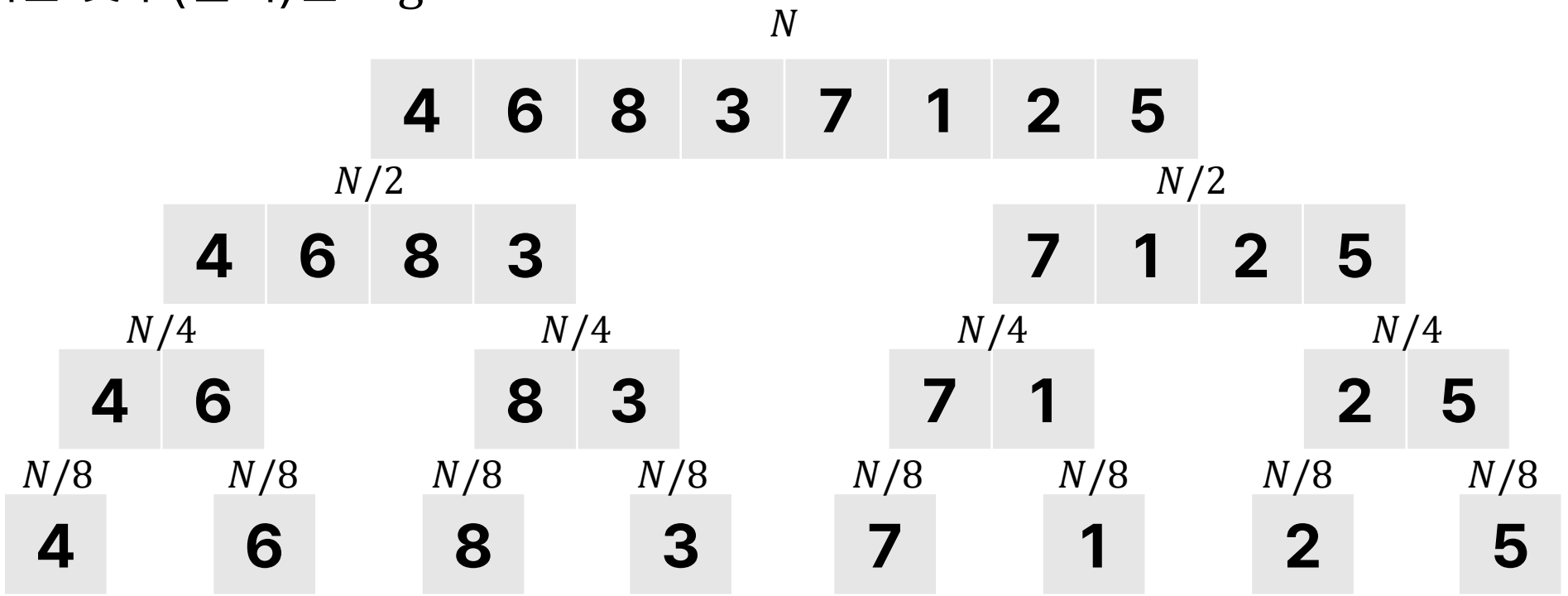
병합 정렬 (Merge Sort) - 결합

병합 정렬 (Merge Sort) - 결합

병합 정렬 (Merge Sort) - 결합

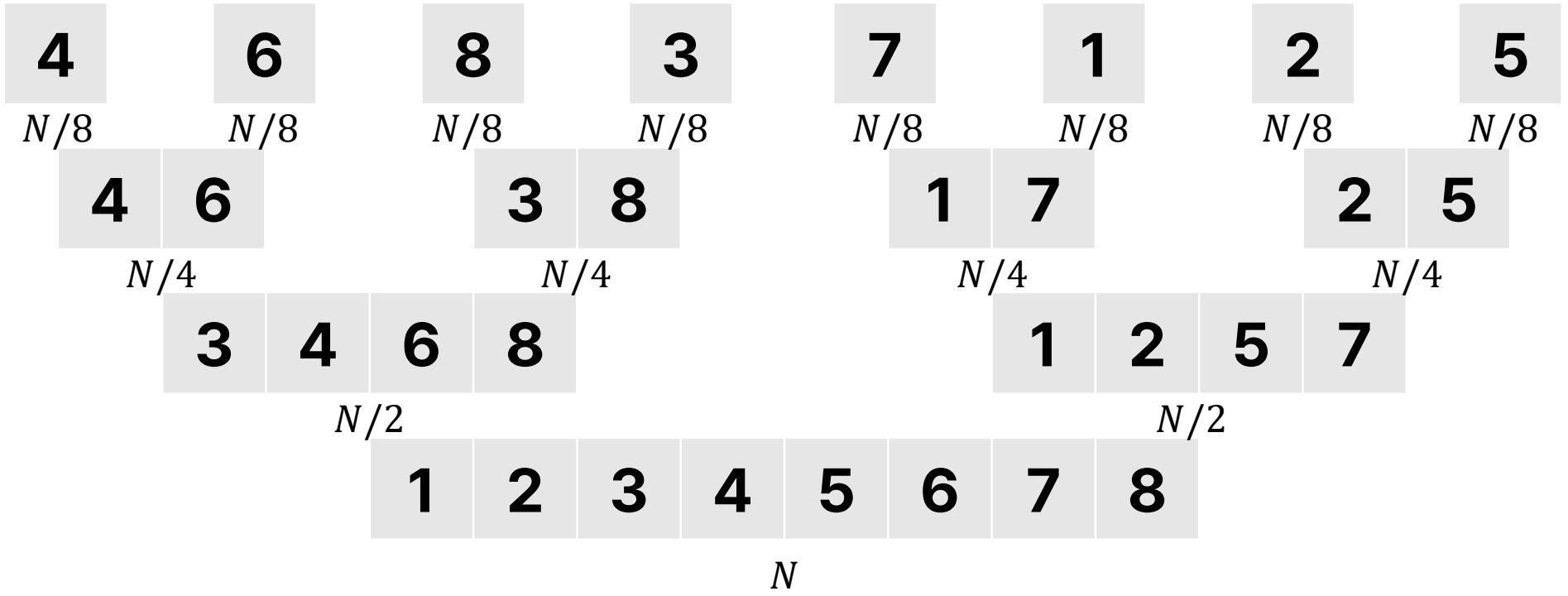
병합 정렬 (Merge Sort) - 분할

- 쪼개는 횟수(깊이)는 $\log N$



병합 정렬 (Merge Sort) - 정복 / 결합

- 합치는 횟수(깊이)는 $\log N$, 합칠 때마다 N개의 원소를 합침



병합 정렬 (Merge Sort) 증명

- $T(n) = 2T\left(\frac{n}{2}\right) + n$ (반으로 쪼개고 둘을 합친다)
- $= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 4T\left(\frac{n}{4}\right) + 2n$
- $= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n = 8T\left(\frac{n}{8}\right) + 3n$
- $\ldots = 2^{\log n} T\left(\frac{n}{2^{\log n}}\right) + \log n \times n$
- $= nT(1) + n\log n = n + n\log n = O(n\log n)$

- PS에서 log는 밑이 2인 로그로 간주
- $T(1) = O(1)$

퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...

퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...

4	6	8	3	7	1	2	5

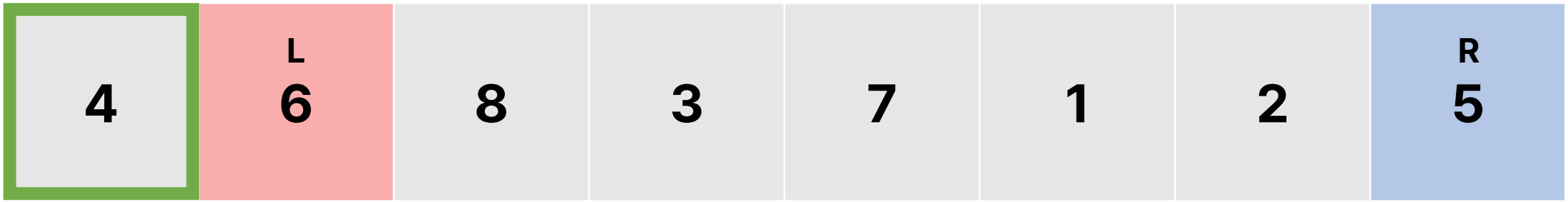
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...

4	L 6	8	3	7	1	2	R 5

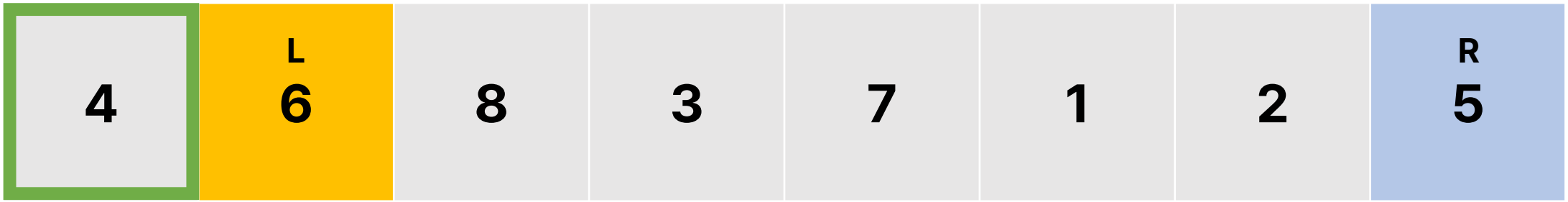
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



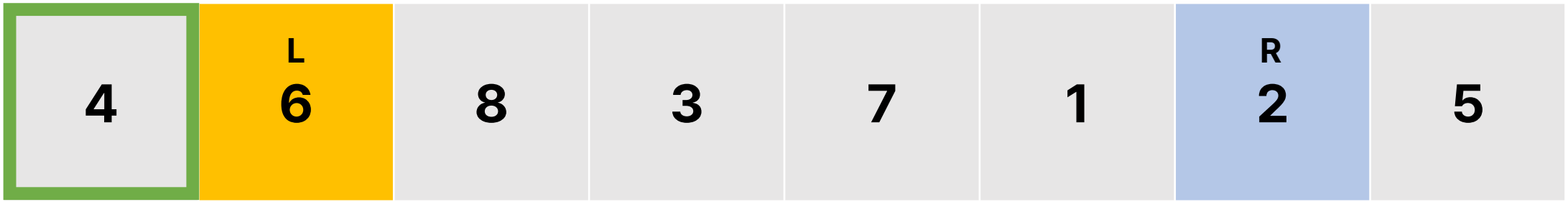
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



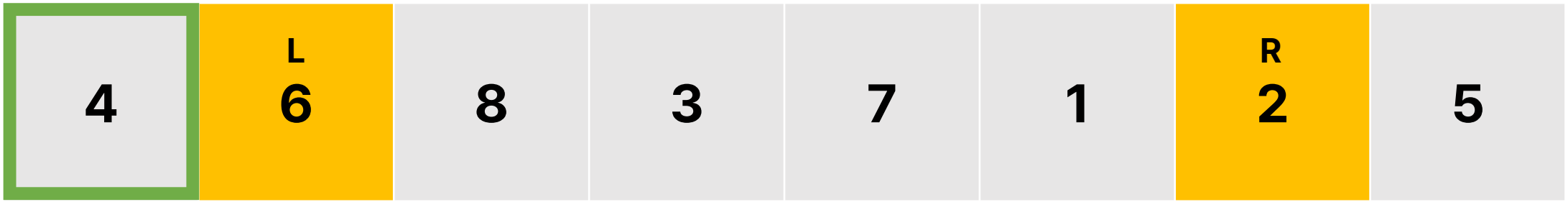
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



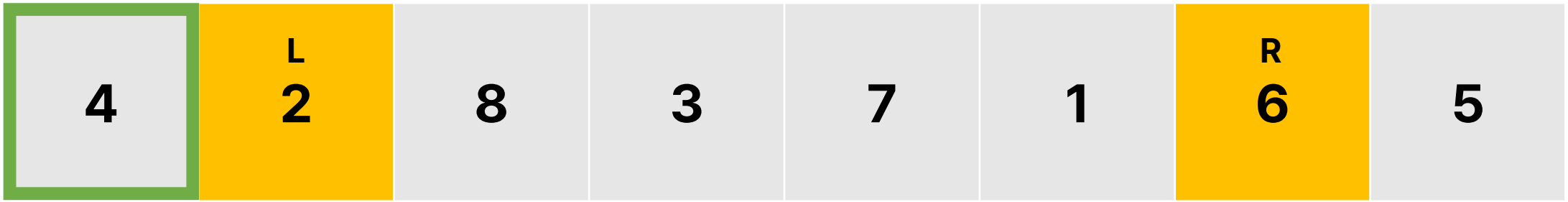
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



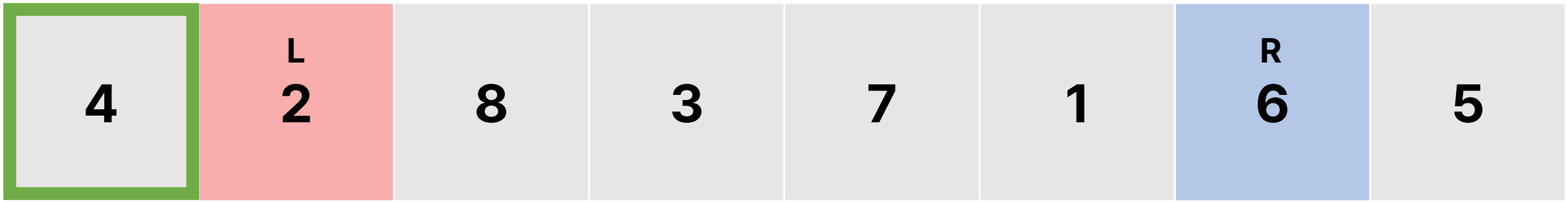
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



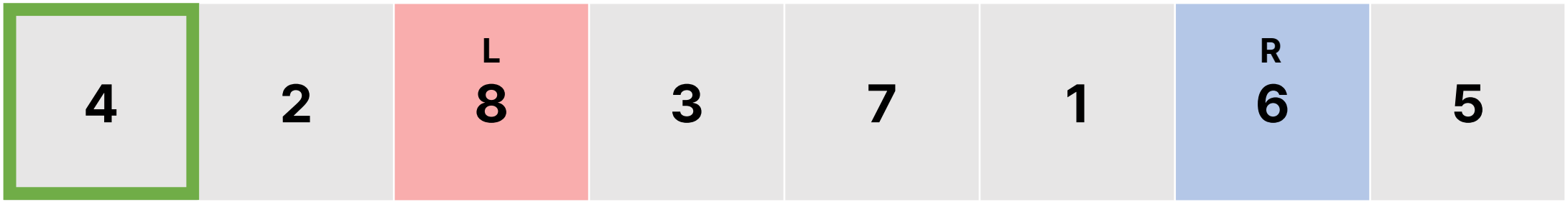
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



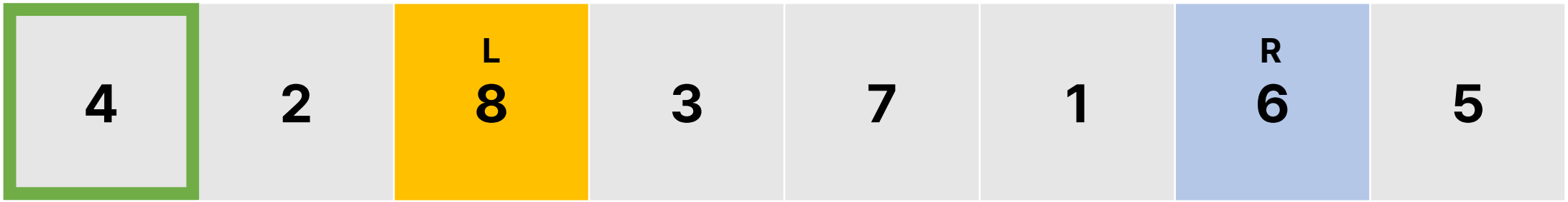
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



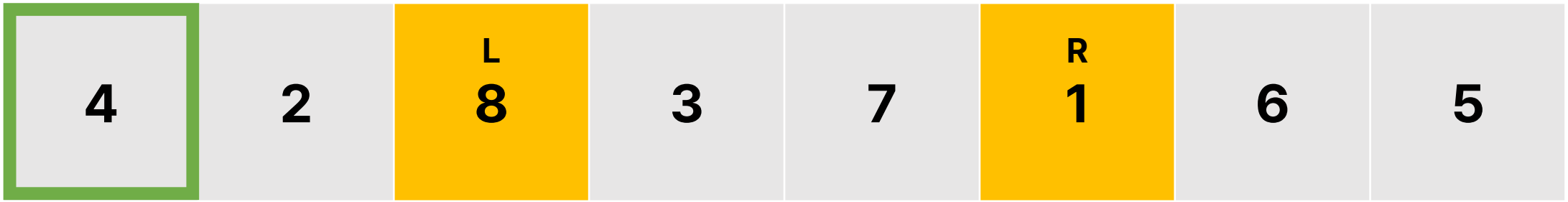
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



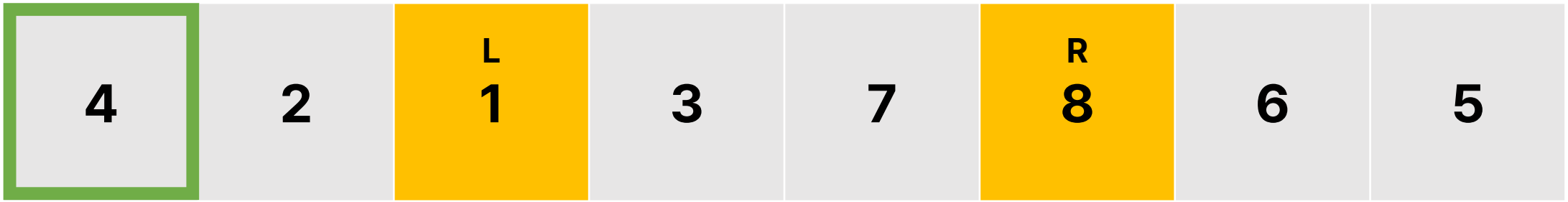
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



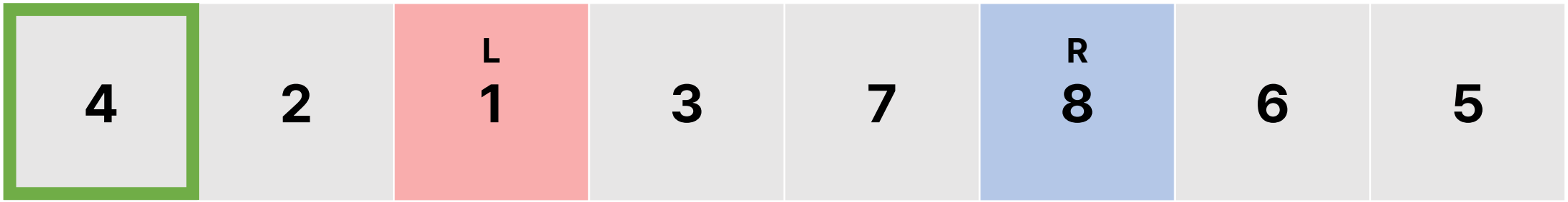
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



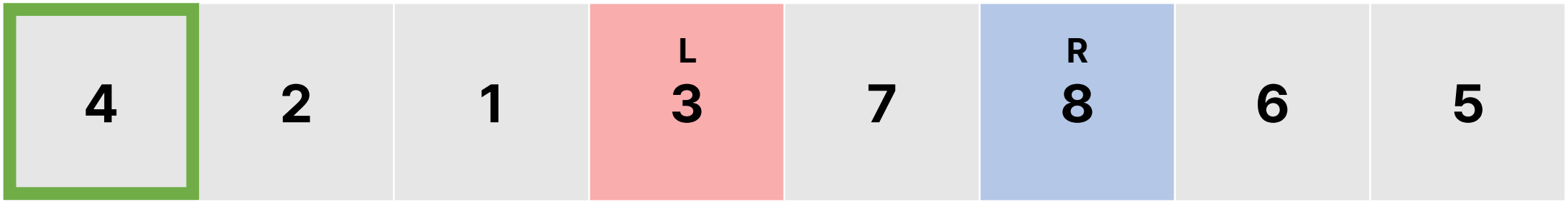
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



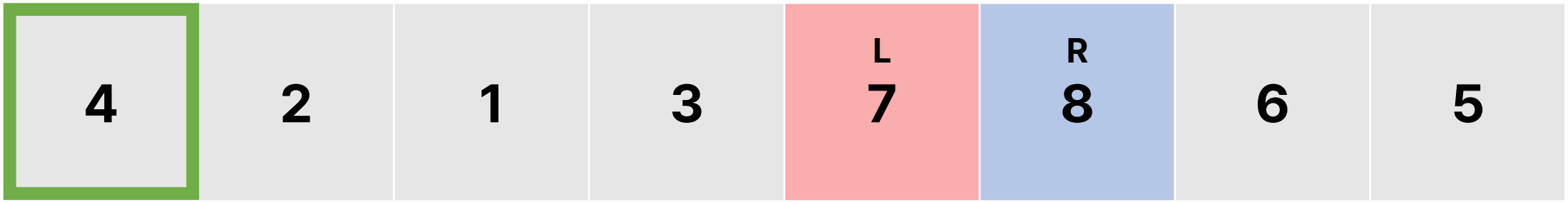
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



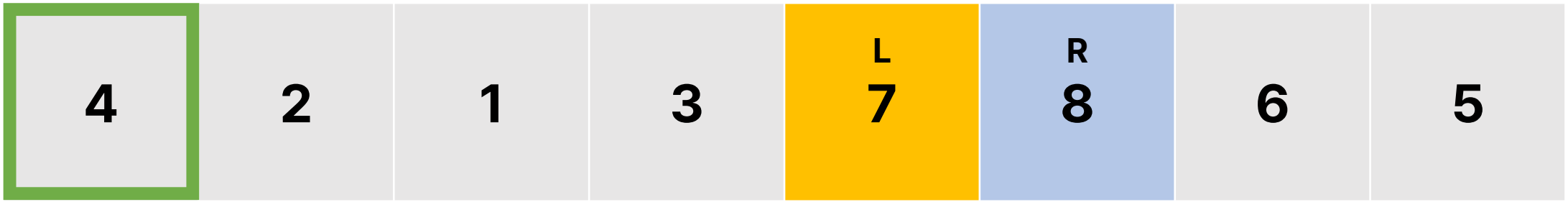
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



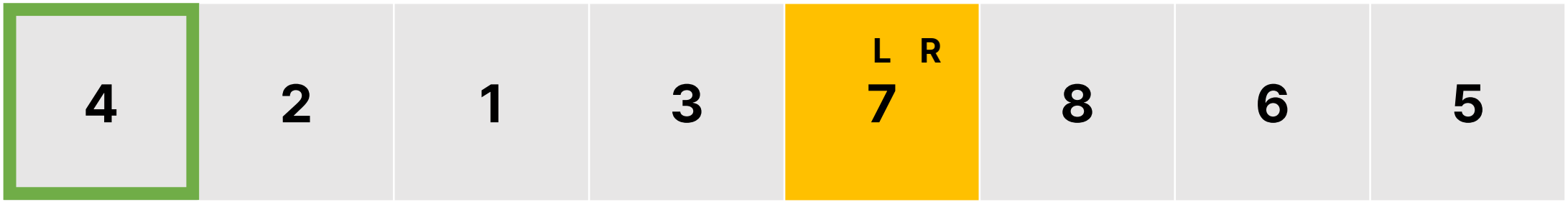
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



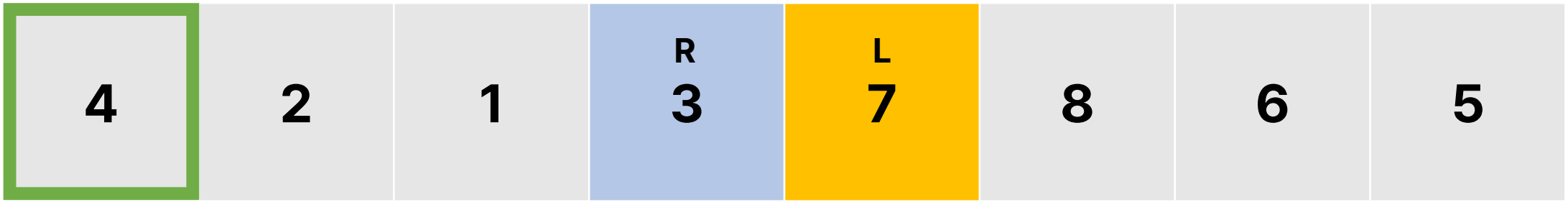
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



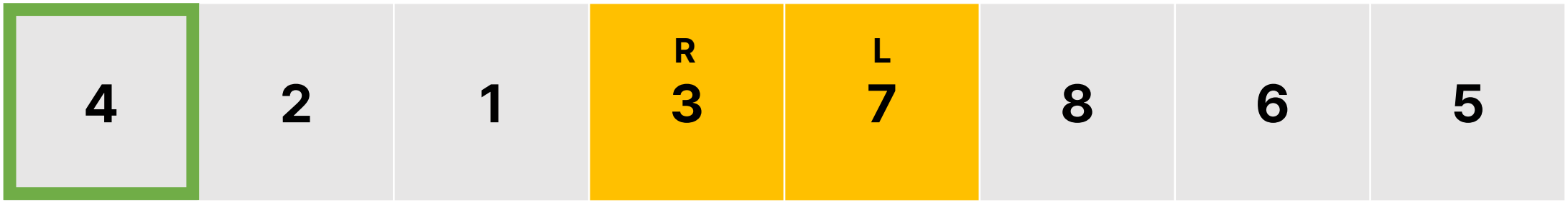
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



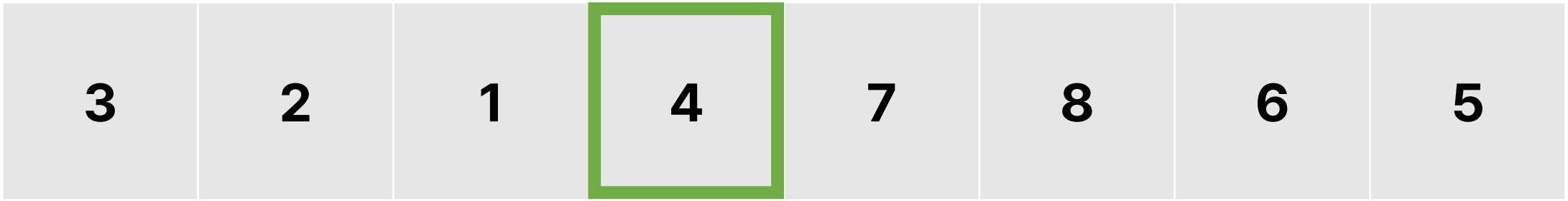
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...



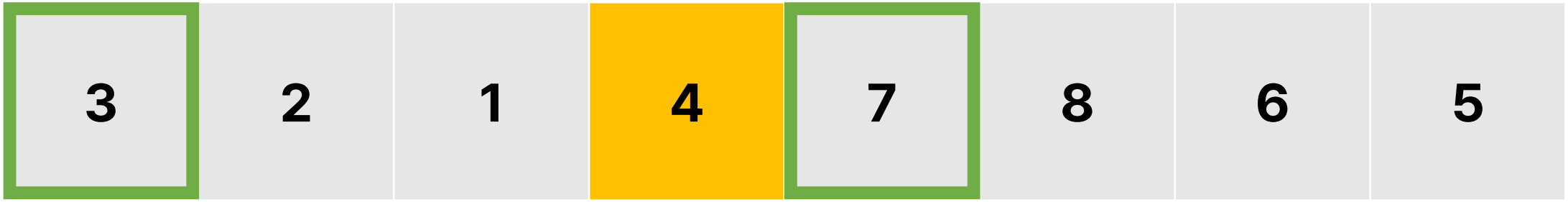
퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...
  - pivot은 자기 자리를 잘 찾아갔고 왼쪽, 오른쪽으로 나누어서 정렬하면 됨



퀵 정렬 (Quick Sort)

- 하나의 기준점(pivot)을 잡고 탐색한다
  - L은 왼쪽에서 오른쪽으로 탐색하다가 pivot보다 큰 원소를 찾으면 멈춘다
  - R은 오른쪽에서 왼쪽으로 탐색하다가 pivot보다 작은 원소를 찾으면 멈춘다
  - L, R이 교차되어 지나가면 멈추고 pivot을 R이 가리키는 원소와 교환한다
  - 지나가지 않았다면 L, R이 가리키는 원소를 교환한다
- pivot을 기준으로 왼쪽과 오른쪽을 나누어 다시 위 행동을 반복한다
  - 나누어진 크기가 0 또는 1이 될 때까지...
  - pivot은 자기 자리를 잘 찾아갔고 왼쪽, 오른쪽으로 나누어서 정렬하면 됨



Code (Merge Sort)

```cpp
// merge sort ========================================
void merge(int s, int mid, int e) {
    int i = s, j = mid + 1, k = s;

    while (i <= mid && j <= e) {
        if (arr[i] <= arr[j]) res[k++] = arr[i++];
        else res[k++] = arr[j++];
    }

    int tmp = i > mid ? j : i;
    while (k <= e) res[k++] = arr[tmp++];
    for (int i = s; i <= e; i++) arr[i] = res[i];
}

void merge_sort(int s, int e) {
    if (s < e) {
        int mid = s + e >> 1;
        merge_sort(s, mid);
        merge_sort(mid + 1, e);
        merge(s, mid, e);
    }
}
// merge sort ========================================
```

```cpp
// merge sort ========================================
void merge_sort_nonRecursion(int s, int e) {
    for (int div = 1; div < n; div *= 2) { // divide size
        for (int left = 0; left + div < n; left += div * 2) { // left side
            int right = left + div;
            int e = right + div;

            if (e > n) e = n;
            int pivot = left, i = left, j = right;
            while (i < right && j < e) {
                if (arr[i] <= arr[j]) res[pivot++] = arr[i++];
                else res[pivot++] = arr[j++];
            }
            while (i < right) res[pivot++] = arr[i++];
            while (j < e) res[pivot++] = arr[j++];

            for (pivot = left; pivot < e; pivot++) arr[pivot] = res[pivot];
        }
    }
}
// merge sort ========================================
```

Code (Quick Sort)

```cpp
// quick sort ==========================================
void quick_sort(int s, int e) {
    if (s >= e) return;
    int pivot = s; // pivot is front
    int i = s + 1;
    int j = e;

    while (i <= j) {
        while (arr[i] <= arr[pivot]) i++;
        while (arr[j] >= arr[pivot] && j > s) j--;

        if (i > j) swap(arr[j], arr[pivot]);
        else swap(arr[i], arr[j]);

        quick_sort(s, j - 1);
        quick_sort(j + 1, e);
    }
}
// quick sort ==========================================
```

```cpp
// quick sort ==========================================
void quick_sort_nonRecursion(int s, int e) {
    stack<int> left, right;
    left.push(s); right.push(e);

    while (!left.empty()) {
        int pivot = s = left.top(); // pivot is front
        int l = s + 1; left.pop();
        int r = e = right.top(); right.pop();

        while (l <= r) {
            while (arr[l] <= arr[pivot]) l++;
            while (arr[r] >= arr[pivot] && r > s) r--;

            if (l > r) swap(arr[r], arr[pivot]);
            else swap(arr[l], arr[r]);

            if (s < r) {
                left.push(s);
                right.push(r);
            }
            if (l < e) {
                left.push(l);
                right.push(e);
            }
        }
    }
}
// quick sort ==========================================
```

수 정렬하기 2 BOJ 2751

- N개의 수가 주어졌을 때, 이를 오름차순으로 정렬하는 프로그램을 작성하시오
- $N(1 \leq N \leq 1\,000\,000)$, 주어지는 수는 절댓값이 $1\,000\,000$보다 작거나 같은 정수

퀵 정렬 (Quick Sort)

- 정렬된 배열에 의해 $O(n^2)$의 시간복잡도를 가짐
  - 따라서 방금 본 문제와 같이 저격 데이터가 있는 경우 시간 초과가 발생
  - 이걸 방지하기 위해 pivot을 랜덤으로 설정하는 방법이 있음
    - 사실상 이 방법을 저격할 수 있는 방법이 없음
  - 적절한 pivot을 $O(n)$에 찾을 수 있도록 해주는 Median of Medians라는 알고리즘도 있음

수 정렬하기 3 BOJ 10989

- N개의 수가 주어졌을 때, 이를 오름차순으로 정렬하는 프로그램을 작성하시오
- $N(1 \leq N \leq 10\ 000\ 000)$, 주어지는 수는 절댓값이 $10\ 000$보다 작거나 같은 정수

Counting Sort

- 정렬하려는 수의 범위가 작음
- 정렬하려는 수의 개수가 많음

- 각 수를 배열 한 칸에 대응시켜 개수를 저장하는 것 또한 일종의 정렬

1	2	3	4	5	6	7	8

Counting Sort

- 정렬하려는 수의 범위가 작음
- 정렬하려는 수의 개수가 많음

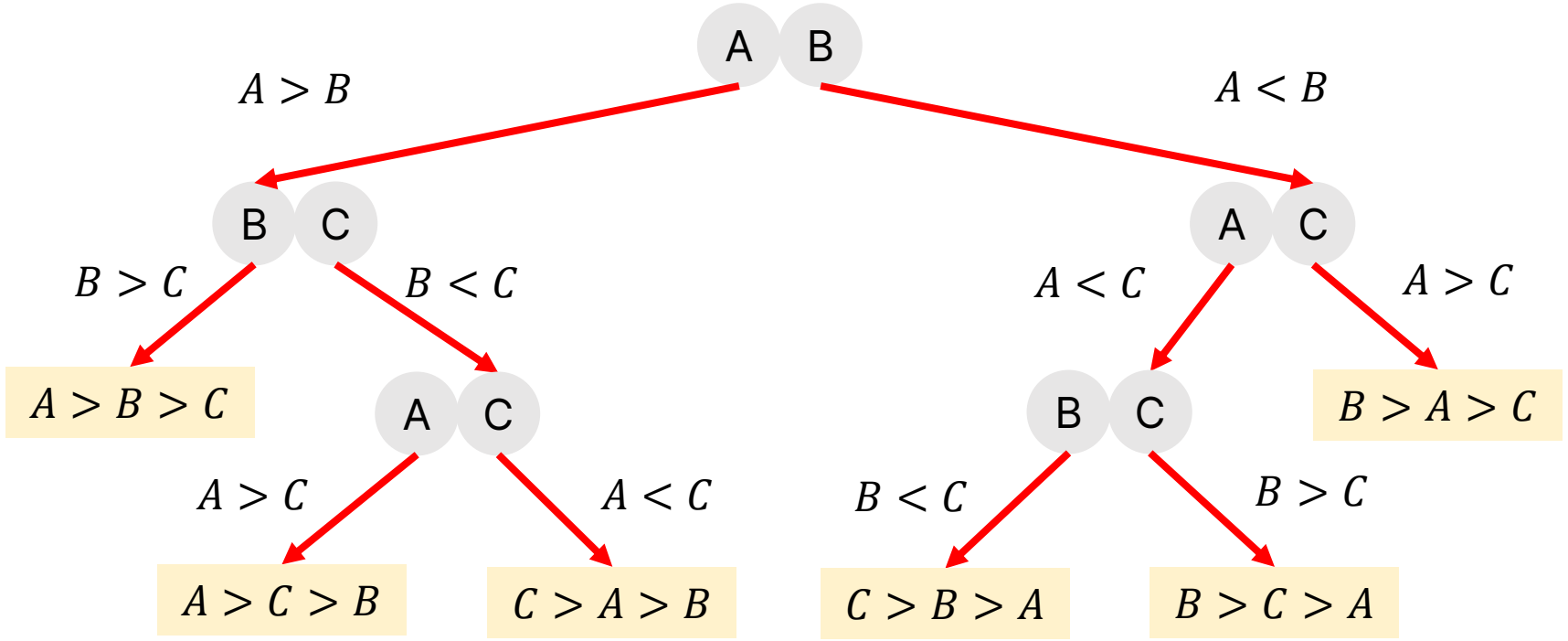
- 각 수를 배열 한 칸에 대응시켜 개수를 저장하는 것 또한 일종의 정렬
- 각 수가 몇 번 나왔는지 세고 나서, 1개 이상 존재하면 개수만큼 출력

15	24	31	40	10	11	18	33
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**

정렬이 $O(n \log n)$보다 빨라질 수 있을까?

- 특수한 정렬의 경우는 그렇다
  - 앞에서 살펴본 Counting Sort가 여기에 속한다
- 기본적으로 정렬을 하기 위해 비교 연산을 한다
  - 정렬하는 경우의 수는 $n!$ 이다 (줄 세우기)
  - 비교 연산으로 만들어지는 decision tree의 높이는 $\log n!$ 이다
    - 두 수를 비교하면 A가 B보다 앞에 있는 경우, B가 A보다 앞에 있는 경우 두 가지로 쪼개지므로 $\log$ 의 형태를 가진다
  - 큰 수 n에 대해 $\log n!$ 과 $n \log n - n$의 비가 1로 수렴한다
    - big-O 표기법을 사용하면 $O(\log n!)$과 $O(n \log n)$이라고 적을 수 있다

정렬이 $O(n \log n)$보다 빨라질 수 있을까?



C++, Python은 어떤 정렬을 사용할까?

- C++은 Intro Sort를 사용한다 (퀵 + 힙 + 삽입)
  - 퀵 정렬을 진행한다
  - 재귀 깊이가 일정 수치를 초과하면 힙 정렬로 전환한다 $(2 \times \log n)$
  - 요소들의 수가 특정 수치 미만이면 삽입 정렬로 전환한다 $(n < 16)$
- Python은 Tim Sort를 사용한다
  - 병합 정렬과 삽입 정렬을 최적화 한 형태이다
  - 배열을 일정 단위로 쪼갠 후 삽입 정렬을 진행한다

C++ sort (STL)

- 헤더 파일 : <algorithm>
- 함수 원형 : void sort(first iterator, last iterator, compare function);
- 시간복잡도 : $O(n \log n)$
- 정렬 범위 : [first, last)

- 첫 N개의 원소를 정렬하는 partial_sort
- stable한 정렬을 지원하는 stable_sort 등도 있다

```cpp
int main(void) {
    fastio;
    vector<int> v = {4, 6, 3, 2, 7, 9, 1, 8, 5};
    sort(v.begin(), v.end());
    for (int x : v)
        cout << x << ' ';
    return 0;
}
```

C++ sort (STL)

- Compare Function
  - 임의의 정렬 기준을 적용하기 위해 사용한다
- 기본적으로 std::greater<T>()와 std::less<T>()가 있다
  - greater – 내림차순 / less – 오름차순 (기본 옵션)

```cpp
struct {
    bool operator()(int a, int b) const { return a < b; }
} cmp;

int main(void) {
    fastio;
    vector<int> v = { 4, 6, 3, 2, 7, 9, 1, 8, 5 };
    sort(v.begin(), v.end(), cmp);
    for (int x : v)
        cout << x << ' ';
    return 0;
}
```

```cpp
struct {
    bool operator()(pair<int, int> a, pair<int, int> b) const {
        if (a.second == b.second) {
            return a.first < b.first;
        } return a.second < b.second;
    }
} cmp;

int main(void) {
    fastio;
    vector<pair<int, int>> v = { {3, 4}, {1, 1}, {1, -1}, {2, 2}, {3, 3} };
    sort(v.begin(), v.end(), cmp);
    for (pair<int, int> x : v)
        cout << x.first << ' ' << x.second << "\n";
    return 0;
}
```

C++ sort (STL)

- bool operator()(T a, T b)
  - a가 b보다 반드시 앞에 나와야 하면 true, 그렇지 않으면 false
  - **Strict Weak Ordering**을 만족해야 함
  - $i < j$에서 compare function이 false를 반환하면 $A[i], A[j]$를 교환한다

- 오름차순: compare(a, b): a < b (a ≤ b가 아님을 주의)
- 내림차순: compare(a, b): a > b (a ≥ b가 아님의 주의)

좌표 정렬하기 BOJ 11650

- 2차원 평면 위의 점 N개가 주어진다.
- 좌표를 x좌표가 증가하는 순으로
  - x좌표가 같으면 y좌표가 증가하는 순서로 정렬한다

좌표 정렬하기 BOJ 11650

```cpp
struct {
  bool operator()(pair<int, int> a, pair<int, int> b) const {
    if (a.first == b.first) {
      return a.second < b.second;
    } return a.first < b.first;
  }
} cmp;

int main(void) {
  fastio;
  vector<pair<int, int>> v = { {3, 4}, {1, 1}, {1, -1}, {2, 2}, {3, 3} };
  sort(v.begin(), v.end(), cmp);
  for (pair<int, int> x : v)
    cout << x.first << ' ' << x.second << "\n";
  return 0;
}
```

Strict Weak Ordering

- 비반사성(irreflexivity)
  - 모든 x에 대해 R(x, x)는 거짓
- 비대칭성(asymmetry)
  - 모든 x, y에 대해 R(x, y)가 참이면 R(y, x)는 거짓
- 추이성(transitivity)
  - 모든 x, y, z에 대해 R(x, y), R(y, z)가 참이면 R(x, z)는 참
- 비비교성의 추이성(transitivity of incomparability)
  - 모든 x, y, z에 대해 R(x, y), R(y, x), R(y, z), R(z, y)가 거짓이면 R(x, z), R(z, x)는 거짓

Strict Weak Ordering – 비교성

- 비교성(comparability)
  - 비교를 할 수 있다
  - (정렬에서) 두 원소의 순서를 정할 수 있다
- 비비교성(incomparability)
  - 비교를 할 수 없다
  - (정렬에서) 두 원소의 순서를 정할 수 없다 (= 두 원소의 우선 순위가 동등)
- ex) 오름차순 정렬
  - 값이 같은 두 원소는 순서를 정할 수 없음
  - 값이 다른 두 원소는 순서를 정할 수 있음

Strict Weak Ordering – 비반사성

- 비반사성(irreflexivity)
  - 모든 x에 대해 R(x, x)는 거짓
    - 모든 x에 대해 x < x는 거짓
  - 값이 같은 두 원소는 비교가 불가능하다
  - compare(x, x)는 두 x중 반드시 먼저 와야 하는 것을 결정할 수 없음

- 오름차순 정렬의 비교 함수로 $a \leq b$를 사용할 수 없는 이유이다

Strict Weak Ordering – 비대칭성

- 비대칭성(asymmetry)
  - 모든 x, y에 대해 R(x, y)가 참이면 R(y, x)는 거짓
    - 모든 x, y에 대해 x < y가 참이면 y < x는 거짓
  - 두 관계가 모두 참이면 두 원소의 순서를 정할 수 없음
  - R(x, y), R(y, x) 모두 거짓이 될 수는 있다

- 사이클이 있는 그래프에서 위상 정렬이 불가능한 이유이다
  - 사이클이 있다면 R(x, y), R(y, x)가 모두 참인 간선이 존재한다는 것

Strict Weak Ordering – 비비교성의 추이성

- 추이성(transitivity)
  - 모든 x, y, z에 대해 R(x, y), R(y, z)가 참이면 R(x, z)는 참
    - 모든 x, y, z에 대해 x < y, y < z이면 x < z는 참
  - 삼단 논법
- 비비교성의 추이성(transitivity of incomparability)
  - 모든 x, y, z에 대해 R(x, y), R(y, x), R(y, z), R(z, y)가 거짓이면 R(x, z), R(z, x)는 거짓
    - 모든 x, y, z에 대해 x == y, y == z이면 x == z 여야 함
  - x, y가 비교 불가능(동등)하고 y, z가 비교 불가능(동등)하면 x, z도 비교 불가능(동등)해야 함

Stable / Unstable Sort

- Stable Sort
  - Insertion Sort
  - Bubble Sort
  - Merge Sort
- Unstable Sort
  - Quick Sort
  - Heap Sort

Stable / Unstable Sort

- 위에서부터 차례대로 가입했다고 하자
- 나이 순서대로 정렬을 해보자

이름	나이
AAA	18
BBB	15
GGG	21
DDD	26
EEE	24
FFF	18
CCC	21

이름	나이
BBB	15
AAA	18
FFF	18
GGG	21
CCC	21
EEE	24
DDD	26

이름	나이
BBB	15
FFF	18
AAA	18
CCC	21
GGG	21
EEE	24
DDD	26

Stable Sort

- Stable Sort는 기존에 있던 순서를 유지한다

이름	나이
AAA	18
BBB	15
GGG	21
DDD	26
EEE	24
FFF	18
CCC	21

이름	나이
BBB	15
AAA	18
FFF	18
GGG	21
CCC	21
EEE	24
DDD	26

이름	나이
BBB	15
FFF	18
AAA	18
CCC	21
GGG	21
EEE	24
DDD	26

Unstable Sort

- Unstable Sort는 기존에 있던 순서가 유지됨이 보장되지 않는다

이름	나이
AAA	18
BBB	15
GGG	21
DDD	26
EEE	24
FFF	18
CCC	21

이름	나이
BBB	15
AAA	18
FFF	18
GGG	21
CCC	21
EEE	24
DDD	26

이름	나이
BBB	15
FFF	18
AAA	18
CCC	21
GGG	21
EEE	24
DDD	26

연습 문제

2750 : 수 정렬하기

2751 : 수 정렬하기 2

10989 : 수 정렬하기 3

11650 : 좌표 정렬하기

References

- https://github.com/justiceHui/Sunrin-SHARC/blob/master/2021-1st/slide/01.pdf