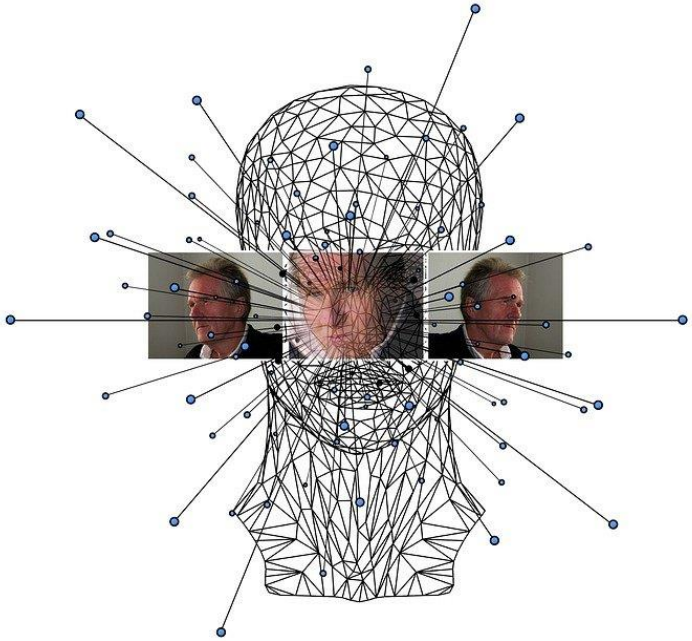
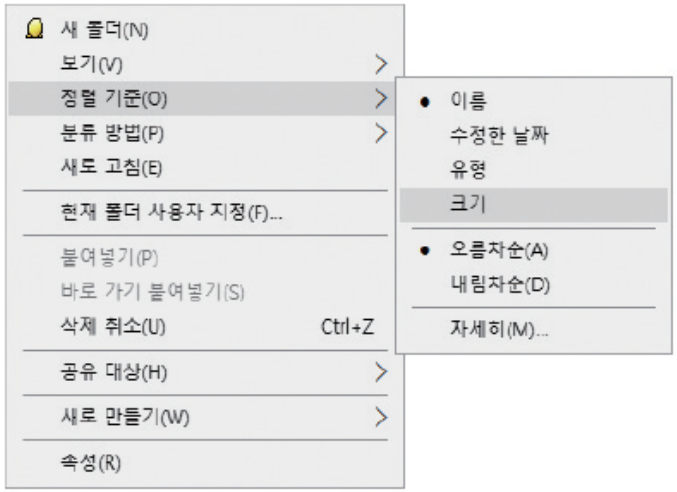


8 정렬

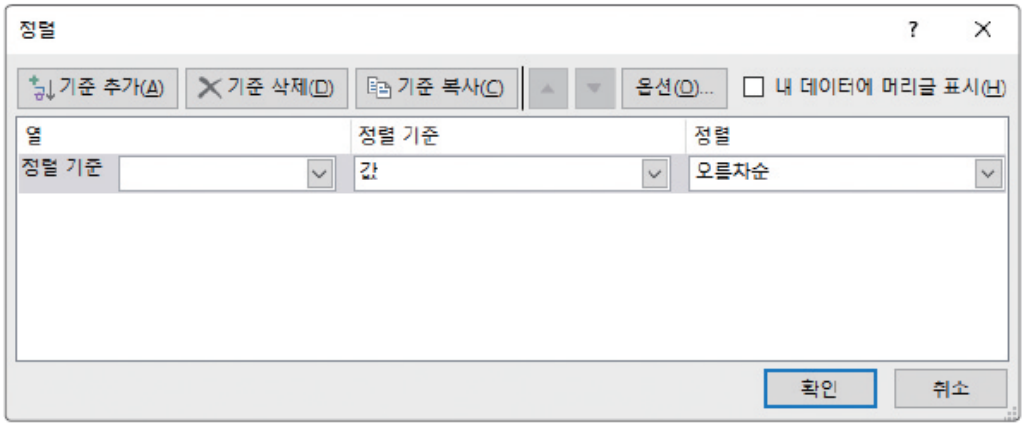
1. 정렬의 이해

❖ 정렬(sort)

- 순서 없이 배열된 자료를 작은 것부터 큰 것 순서인 오름차순Ascending이나 큰 것부터 작은 것 순서인 내림차순Descending으로 재배열하는 것
- 키(Key) - 자료를 정렬하는데 사용하는 기준이 되는 특정 값
- 정렬의 예



(a) 아이콘 정렬하기



(b) 데이터 시트에서 정렬하는 기준 정하기

1. 정렬의 이해

(c) 쇼핑몰에서 제품 정렬하기

그림 9-1 정렬의 예

1. 정렬의 이해

❖ 정렬 방식의 분류

표 9-1 정렬 방식의 분류

기준	정렬 방식	설명
실행 방법	비교식 정렬 Comparative Sort	비교할 각 키값을 한 번에 두 개씩 비교하여 교환함으로써 정렬을 실행하는 방식
	분배식 정렬 Distribute Sort	키값을 기준으로 하여 자료를 여러 개의 부분집합으로 분해하고, 각 부분집합을 정렬함으로써 전체를 정렬하는 방식
정렬 장소	내부 정렬 Internal Sort	컴퓨터 메모리 내부에서 정렬
	외부 정렬 External Sort	메모리의 외부인 보조 기억 장치에서 정렬

1. 정렬의 이해

- 내부 정렬(internal sort)
  - 정렬할 자료를 메인 메모리에 올려서 정렬하는 방식
  - 정렬 속도가 빠르지만 정렬할 수 있는 자료의 양이 메인 메모리의 용량에 따라 제한됨

표 9-2 내부 정렬의 분류

구분	종류	설명
비교식	교환 방식	키를 비교하고 교환하여 정렬하는 방식(선택 정렬, 버블 정렬, 퀵 정렬)
	삽입 방식	키를 비교하고 삽입하여 정렬하는 방식(삽입 정렬, 셸 정렬)
	병합 방식	키를 비교하고 병합하여 정렬하는 방식(2-way 병합, n-way 병합)
	선택 방식	이진 트리를 사용하여 정렬하는 방식(히프 정렬, 트리 정렬)
분배식	분배 방식	키를 구성하는 값을 여러 개의 부분집합에 분배하여 정렬하는 방식(기수 정렬)

- 외부 정렬(external sort)
  - 정렬할 자료를 보조 기억장치에서 정렬하는 방식
  - 대용량의 보조 기억 장치를 사용하기 때문에 내부 정렬보다 속도는 떨어지지만 내부 정렬로 처리할 수 없는 대용량의 자료에 대한 정렬 가능
- 외부 정렬 방식
  - 병합 방식 : 파일을 부분 파일로 분리하여 각각을 내부 정렬 방법으로 정렬하여 병합하는 정렬 방식 (2-way 병합, n-way 병합)

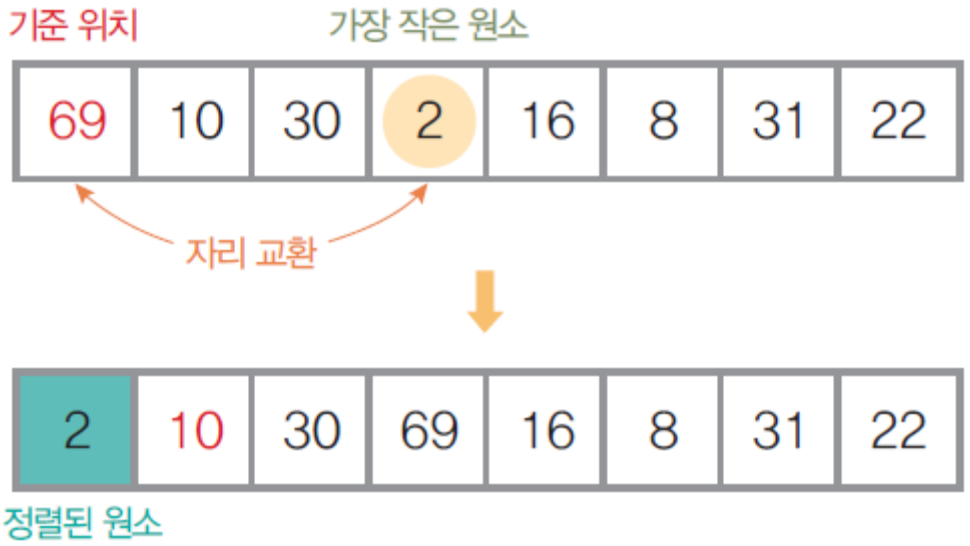
2. 선택 정렬

❖ 선택 정렬(selection sort)

- 전체 원소들 중에서 기준 위치에 맞는 원소를 선택하여 자리를 교환하는 방식으로 정렬
- 수행 방법
  - 전체 원소 중에서 가장 작은 원소를 찾아 첫 번째 원소와 자리를 교환
  - 두 번째로 작은 원소를 찾아 선택하여 두 번째 원소와 자리를 교환
  - 세 번째로 작은 원소를 찾아 선택하여 세 번째 원소와 자리를 교환
  - 이 과정을 반복하면서 정렬을 완성

2. 선택 정렬

- 정렬되지 않은 {69, 10, 30, 2, 16, 8, 31, 22}의 자료를 선택 정렬 방법으로 정렬하는 과정
  - ① 1단계 : 첫째 자리를 기준 위치로 정하고, 전체 중 가장 작은 원소 2를 선택한 후 기준 위치에 있는 원소 69와 자리를 교환



2. 선택 정렬

② 2단계 : 둘째 자리를 기준 위치로 정하고, 나머지 원소 중에서 가장 작은 원소인 8을 선택한 후 기준 위치에 있는 원소 10과 자리를 교환

2. 선택 정렬

③ 3단계 : 셋째 자리를 기준 위치로 정하고, 나머지 원소 중 가장 작은 원소인 10을 선택한 후 기준 위치에 있는 원소 30과 자리를 교환

2. 선택 정렬

④ 4단계 : 넷째 자리를 기준 위치로 정하고, 나머지 원소 중 가장 작은 원소인 16을 선택한 후 기준 위치에 있는 원소 69와 자리를 교환

2. 선택 정렬

⑤ 5단계 : 다섯째 자리를 기준 위치로 정하고, 나머지 원소 중에서 가장 작은 원소 22를 선택한 후 기준위치에 있는 원소 69와 자리를 교환

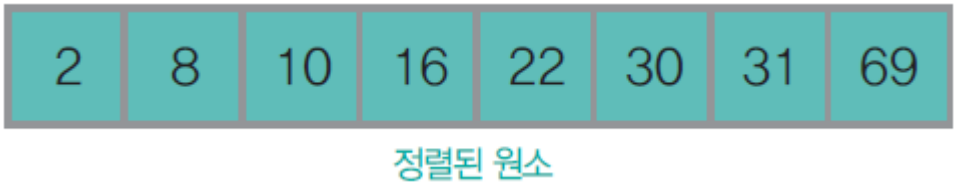
2. 선택 정렬

⑥ 6단계 : 여섯째 자리를 기준 위치로 정하고, 나머지 원소 중에서 가장 작은 원소 30을 선택한 후 기준위치에 있는 원소 30과 자리를 교환(제자리)

2. 선택 정렬

⑦ 7단계 : 일곱째 자리를 기준 위치로 정하고, 나머지 원소 중에서 가장 작은 원소 31을 선택한 후 기준 위치에 있는 원소 31과 자리를 교환(제자리)

⑧ 마지막에 남은 원소는 전체 원소 중에서 가장 큰 원소로, 마지막 자리에 남아 이미 정렬된 상태가 되므로 실행을 종료



2. 선택 정렬

❖ 선택 정렬 알고리즘

- 크기가 n인 배열 a[ ]의 원소를 선택 정렬하는 알고리즘

알고리즘 9-1 선택 정렬

```
selectionSort(a[], n)
    for (i←1; i<n; i←i + 1) do {
        a[i], …, a[n - 1] 중에서 가장 작은 원소 a[k]를 선택해 a[i]와 교환한다.
    }
end selectionSort()
```

2. 선택 정렬

- 메모리 사용공간
  - n개의 원소에 대하여 n개의 메모리 사용
- 비교횟수
  - 1단계 : 첫 번째 원소를 기준으로 n개의 원소 비교
    2단계 : 두 번째 원소를 기준으로 마지막 원소까지 n-1개의 원소 비교
    3단계 : 세 번째 원소를 기준으로 마지막 원소까지 n-2개의 원소 비교
    i 단계 : i 번째 원소를 기준으로 n-i개의 원소 비교

    전체 비교횟수 : (n-1)+(n-2)+ … +2+1 = $\sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2}$

- 어떤 경우에서나 비교횟수가 같으므로 시간 복잡도는 $O(n^2)$이 됨

3. 버블 정렬

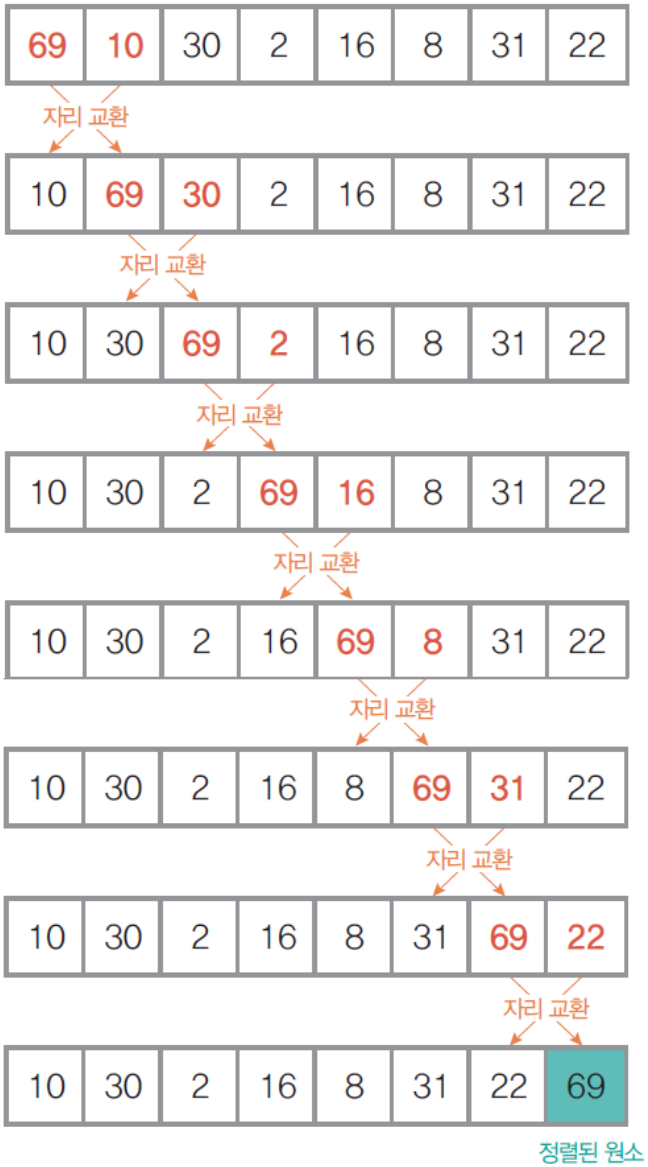
❖ 버블 정렬bubble sort의 이해

- 인접한 두 개의 원소를 비교하여 자리를 교환하는 방식
  - 첫 번째 원소부터 마지막 원소까지 반복하여 한 단계가 끝나면 가장 큰 원소가 마지막 자리로 정렬
  - 첫 번째 원소부터 인접한 원소끼리 계속 자리를 교환하면서 맨 마지막 자리로 이동하는 모습이 물 속에서 물 위로 올라오는 물방울 모양과 같다고 하여 버블(bubble) 정렬이라 함.

- 정렬되지 않은 {69, 10, 30, 2, 16, 8, 31, 22}의 자료들을 버블 정렬 방법으로 정렬하는 과정

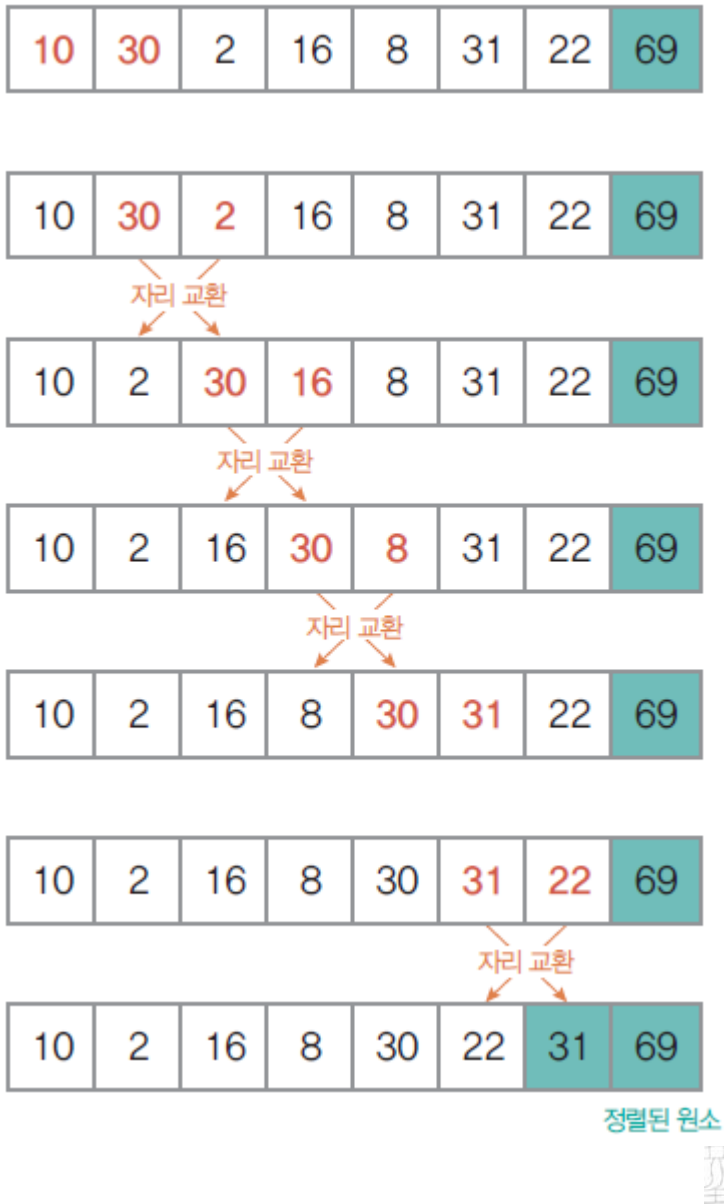
  ① 1단계 : 인접한 두 원소를 비교하여 자리를 교환하는 작업을 첫 번째 원소부터 마지막 원소까지 차례로 반복하여 가장 큰 원소 69를 마지막 자리로 정렬.

3. 버블 정렬


69 10 30 2 16 8 31 22
자리 교환
10 69 30 2 16 8 31 22
자리 교환
10 30 69 2 16 8 31 22
자리 교환
10 30 2 69 16 8 31 22
자리 교환
10 30 2 16 69 8 31 22
자리 교환
10 30 2 16 8 69 31 22
자리 교환
10 30 2 16 8 31 69 22
자리 교환
10 30 2 16 8 31 22 69
정렬된 원소

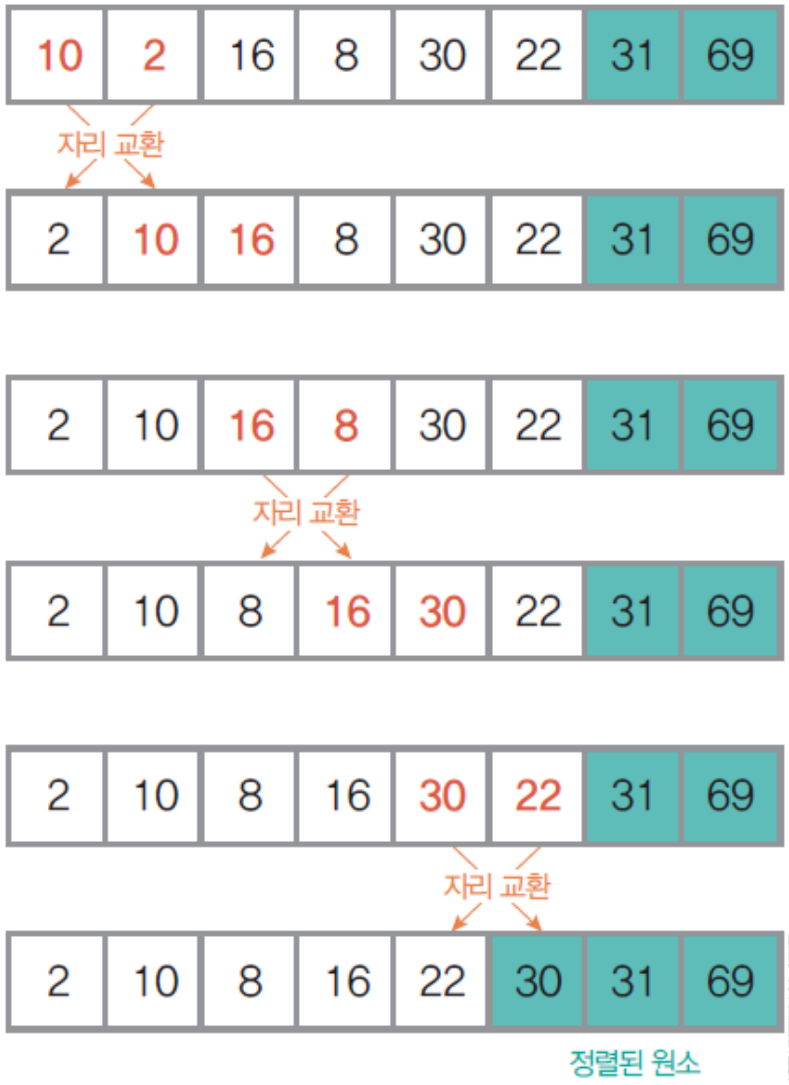
3. 버블 정렬

② 2단계 : 정렬하지 않은 원소에 대해 두 번째 버블 정렬을 수행하면, 나머지 원소 중에서 가장 큰 원소 31이 끝에서 둘째 자리로 정렬



3. 버블 정렬

③ 3단계 : 세 번째 버블 정렬을 수행하면, 나머지 원소 중에서 가장 큰 원소 30이 끝에서 셋째 자리로 정렬



3. 버블 정렬

④ 4단계 : 네 번째 버블 정렬을 정렬하지 않은 원소에 대해 수행하면, 나머지 원소 중에서 가장 큰 원소22가 끝에서 넷째 자리로 정렬

3. 버블 정렬

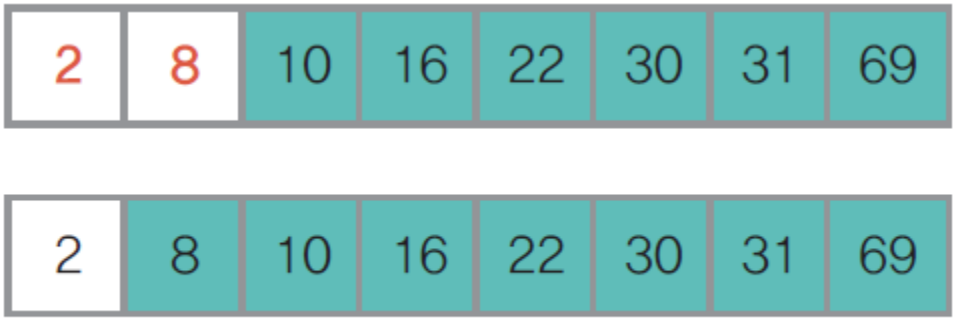
⑤ 5단계 : 다섯 번째 버블 정렬을 수행하면, 나머지 원소 중에서 가장 큰 원소 16이 끝에서 다섯째 자리로 정렬

3. 버블 정렬

⑥ 6단계 : 여섯 번째 버블 정렬을 수행하면, 나머지 원소 중에서 가장 큰 원소 10이 끝에서 여섯째 자리로 정렬

3. 버블 정렬

⑦ 7단계 : 나머지 원소 중에서 가장 큰 원소 8이 끝에서 일곱째 자리로 정렬



⑧ 마지막에 남은 첫째 원소는 전체 원소 중에서 가장 작은 원소로, 가장 앞자리에 남아 이미 정렬된 상태가 되므로 실행을 종료

3. 버블 정렬

❖ 버블 정렬 알고리즘

- 크기가 n인 배열 a[ ]의 원소를 버블 정렬하는 알고리즘

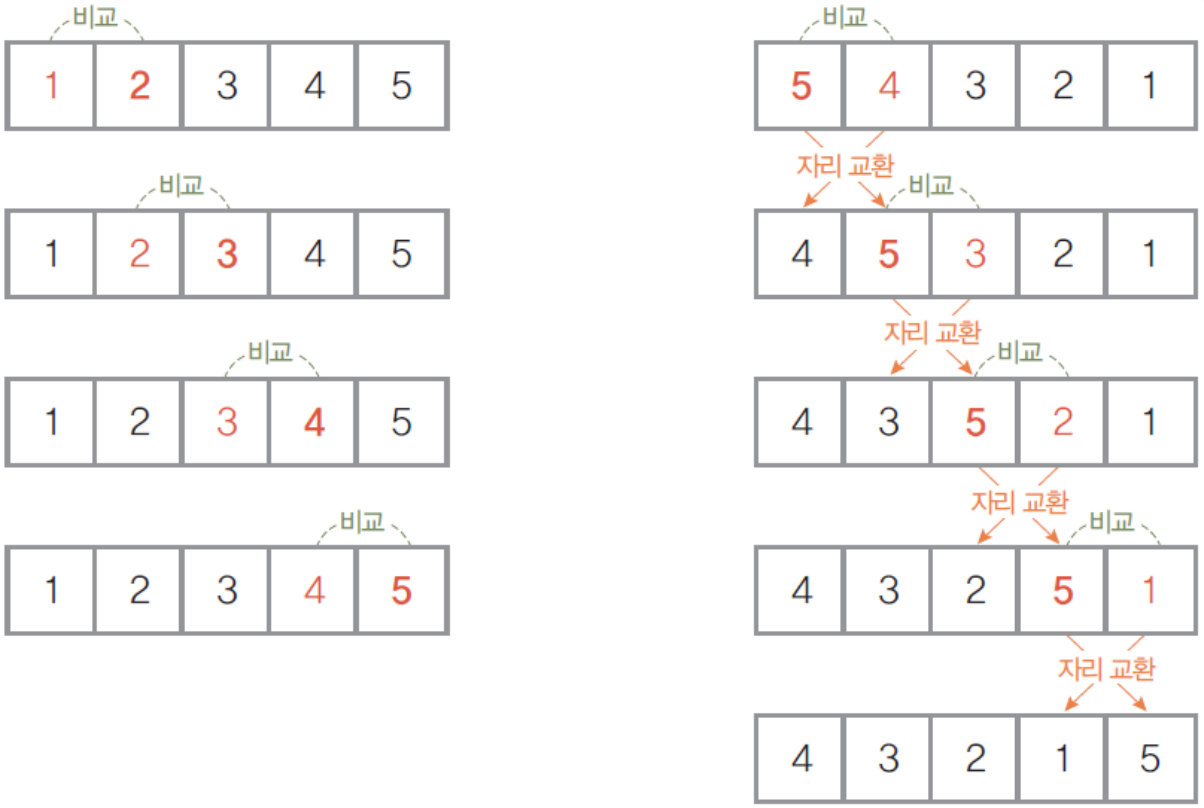
알고리즘 9-2 버블 정렬

```
bubbleSort(a[], n)
    for (i ← n - 1; i≥0; i ← i - 1) do {
        for (j ← 0; j<i; j ← j + 1) do {
            if (a[j] > a[j + 1]) then {
                temp ← a[j];
                a[j] ← a[j + 1];
                a[j + 1] ← temp;
            }
        }
    }
end bubbleSort()
```

3. 버블 정렬

- 메모리 사용공간
  - n개의 원소에 대하여 n개의 메모리 사용
- 연산 시간
  - 최선의 경우 : 자료가 이미 정렬되어있는 경우
    - 비교횟수 : i번째 원소를 (n-i)번 비교하므로, $\frac{n(n-1)}{2}$ 번
    - 자리교환횟수 : 자리교환이 발생하지 않음
  - 최악의 경우 : 자료가 역순으로 정렬되어있는 경우
    - 비교횟수 : i번째 원소를 (n-i)번 비교하므로, $\frac{n(n-1)}{2}$ 번
    - 자리교환횟수 : i번째 원소를 (n-i)번 교환하므로, $\frac{n(n-1)}{2}$ 번
  - 최선의 경우와 최악의 경우에 대한 평균 시간 복잡도를 빅-오 Big Oh 표기법으로 나타내면 $O(n^2)$이 됨

3. 버블 정렬



- 전체 비교 횟수 : $1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$
- 전체 자리 교환 횟수 : 0

  → 시간 복잡도 : $O(n^2)$

(a) 순차 정렬된 경우 : 최선의 경우

- 전체 비교 횟수 : $1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$
- 전체 자리 교환 횟수 : $1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$

  → 시간 복잡도 : $O(n^2)$

(b) 역순 정렬된 경우 : 최악의 경우

그림 9-2 순차 정렬된 경우와 역순 정렬된 경우의 버블 정렬 비교

4. 퀵 정렬

❖ 퀵 정렬(quick sort)의 이해

- 정렬할 전체 원소에 대해서 정렬을 수행하지 않고, 기준 값을 중심으로 왼쪽 부분 집합과 오른쪽 부분 집합으로 분할하여 정렬하는 방법
  - 왼쪽 부분 집합에는 기준 값보다 작은 원소들을 이동시키고, 오른쪽 부분 집합에는 기준 값보다 큰 원소들을 이동시킴
  - 기준 값 : 피봇(pivot)
    - 일반적으로 전체 원소 중에서 가운데에 위치한 원소를 선택
- 퀵 정렬은 다음의 두 가지 기본 작업을 반복 수행하여 완성
  - 분할(divide)
    - 정렬할 자료들을 기준값을 중심으로 두 개로 나눠 부분집합을 만듦
  - 정복(conquer)
    - 부분집합 안에서 기준값의 정렬 위치를 정함

4. 퀵 정렬

- 퀵 정렬 동작 규칙

❶ 왼쪽 끝에서 오른쪽으로 움직이면서 크기를 비교하여 피봇보다 크거나 같은 원소를 찾아 L로 표시한다. 단, L은 R과 만나면 더 이상 오른쪽으로 이동하지 못하고 멈춘다.

❷ 오른쪽 끝에서 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾아 R로 표시한다. 단, R은 L과 만나면 더 이상 왼쪽으로 이동하지 못하고 멈춘다.

❸-ⓐ ❶과 ❷에서 찾은 L 원소와 R 원소가 있는 경우, 서로 교환하고 L과 R의 현재 위치에서 ❶과 ❷ 작업을 다시 수행한다.

❸-ⓑ ❶~❷를 수행하면서 L과 R이 같은 원소에서 만나 멈춘 경우, 피봇과 R의 원소를 서로 교환한다. 교환된 자리를 피봇 위치로 확정하고 현재 단계의 퀵 정렬을 끝낸다.

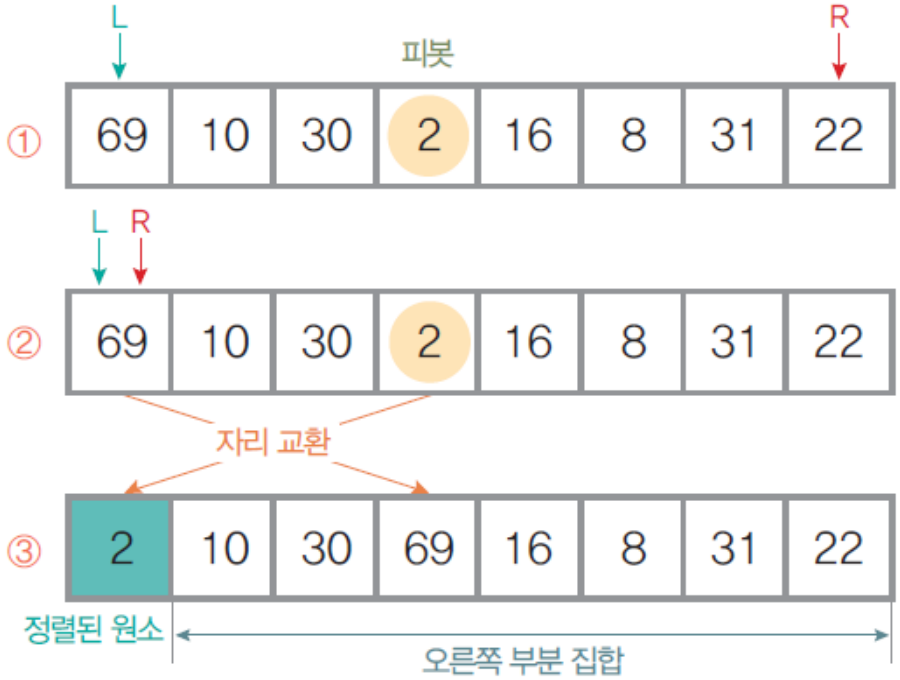
❹ 피봇의 확정된 위치를 기준으로 만들어진 새로운 왼쪽 부분집합과 오른쪽 부분집합에 대해서 ❶~❸의 퀵 정렬을 순환적으로 반복 수행하는데, 모든 부분집합의 크기가 1 이하가 되면 전체 퀵 정렬을 종료한다.

그림 9-3 퀵 정렬을 위한 동작 규칙

4. 퀵 정렬

- 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 퀵 정렬 방법으로 정렬하는 과정 (가운데 원소를 피봇으로 정함)

  ⑴ 1단계 : ① 원소 2를 피봇으로 선택하고 퀵 정렬을 시작 ② L은 정렬 범위의 왼쪽 끝에서 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를 찾고, R은 정렬 범위의 오른쪽 끝에서 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음. L은 원소 69를 찾았지만, R은 피봇보다 작은 원소를 찾지 못한 상태로 원소 69에서 L과 만남. L과 R이 만나 더 이상 진행할 수 없는 상태가 되었으므로,③ 원소 69를 피봇과 자리를 교환하고 피봇 원소 2의 위치를 확정



4. 퀵 정렬

⑵ 2단계 : 위치가 확정된 피봇 2의 왼쪽 부분집합은 공집합이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합에 대해서 퀵 정렬을 수행.

① 오른쪽 부분집합의 원소가 일곱 개이므로 가운데 있는 원소 16을 피봇으로 선택하고 퀵 정렬을 시작.

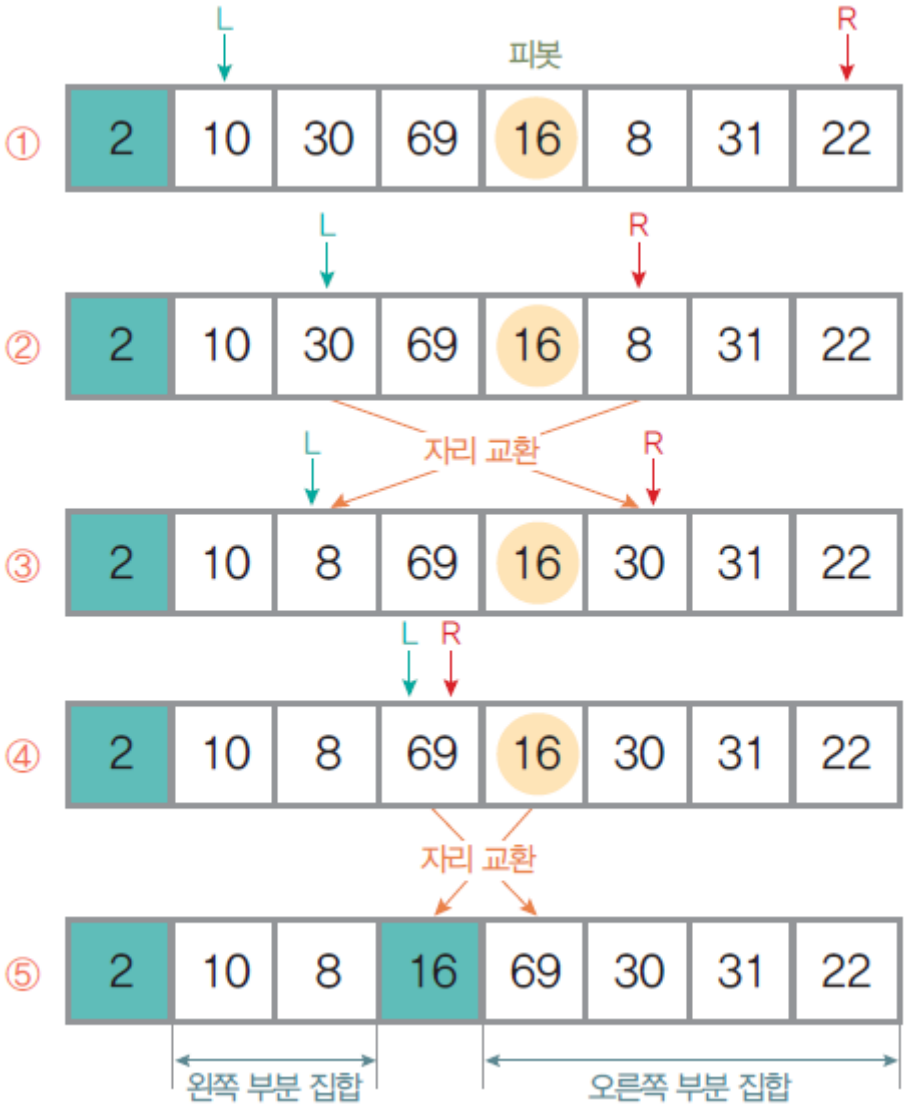
② L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소인 30을 찾고, R은 왼쪽으로 움직이면서 피봇보다 작은 원소인 8을 찾음.

③ L이 찾은 30과 R이 찾은 8을 서로 자리를 교환한 후 현재 위치에서 L은 다시 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소69를 찾고 R은 피봇보다 작은 원소를 찾음.

④ R이 원소 69에서 L과 만나 더 이상 진행할 수 없는 상태가 되었으므로,

⑤ 원소 69를 피봇과 교환하고 피봇 원소 16의 위치를 확정

4. 퀵 정렬


L
피봇
R
①
2 10 30 69 16 8 31 22
L
R
②
2 10 30 69 16 8 31 22
L
자리 교환
R
③
2 10 8 69 16 30 31 22
L R
④
2 10 8 69 16 30 31 22
자리 교환
⑤
2 10 8 16 69 30 31 22
왼쪽 부분 집합
오른쪽 부분 집합

4. 퀵 정렬

⑶ 3단계 : 위치가 확정된 피봇 16을 기준으로 새로 생긴 왼쪽 부분집합에서 퀵 정렬을 수행

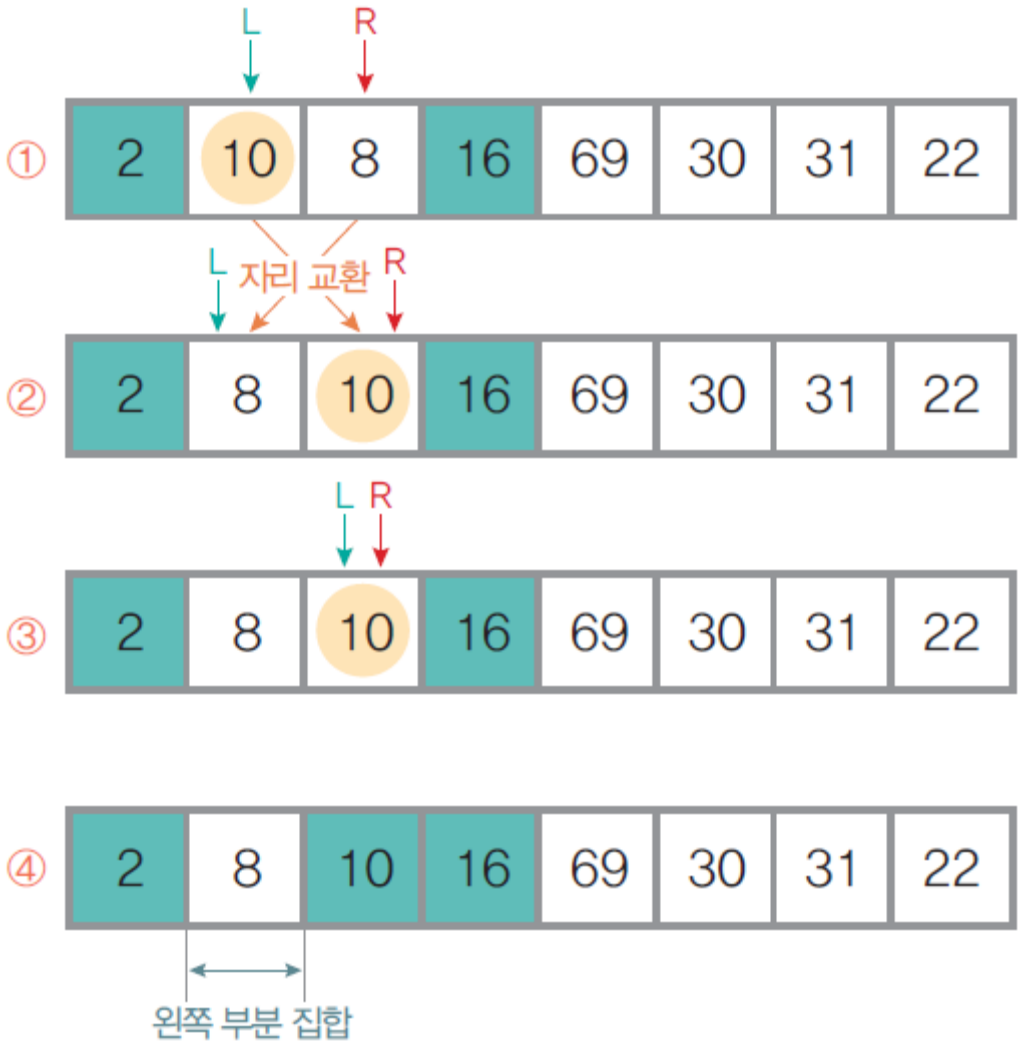
① 원소 10을 피봇으로 선택하고 L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를, R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음

② L이 찾은 10과 R이 찾은 8을 서로 교환

③ 현재 위치에서 L은 다시 오른쪽으로 움직이다가 원소 10에서 R과 만나서 멈추게 됨. 더 이상 진행할 수 없는 상태가 되었으므로,

④ R의 원소와 피봇을 교환하고 피봇 원소 10의 위치를 확정한다(이 경우에는 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음)

4. 퀵 정렬


L
R
①
2 10 8 16 69 30 31 22
L 자리 교환 R
②
2 8 10 16 69 30 31 22
L R
③
2 8 10 16 69 30 31 22
④
2 8 10 16 69 30 31 22
왼쪽 부분 집합

4. 퀵 정렬

⑷ 4단계 : 피봇 10의 왼쪽 부분집합은 원소가 한 개이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합은 공집합이므로 역시 퀵 정렬을 수행하지 않음
이제 [2]에서 피봇이었던 원소 16의 오른쪽 부분집합에 대해서 퀵 정렬을 수행

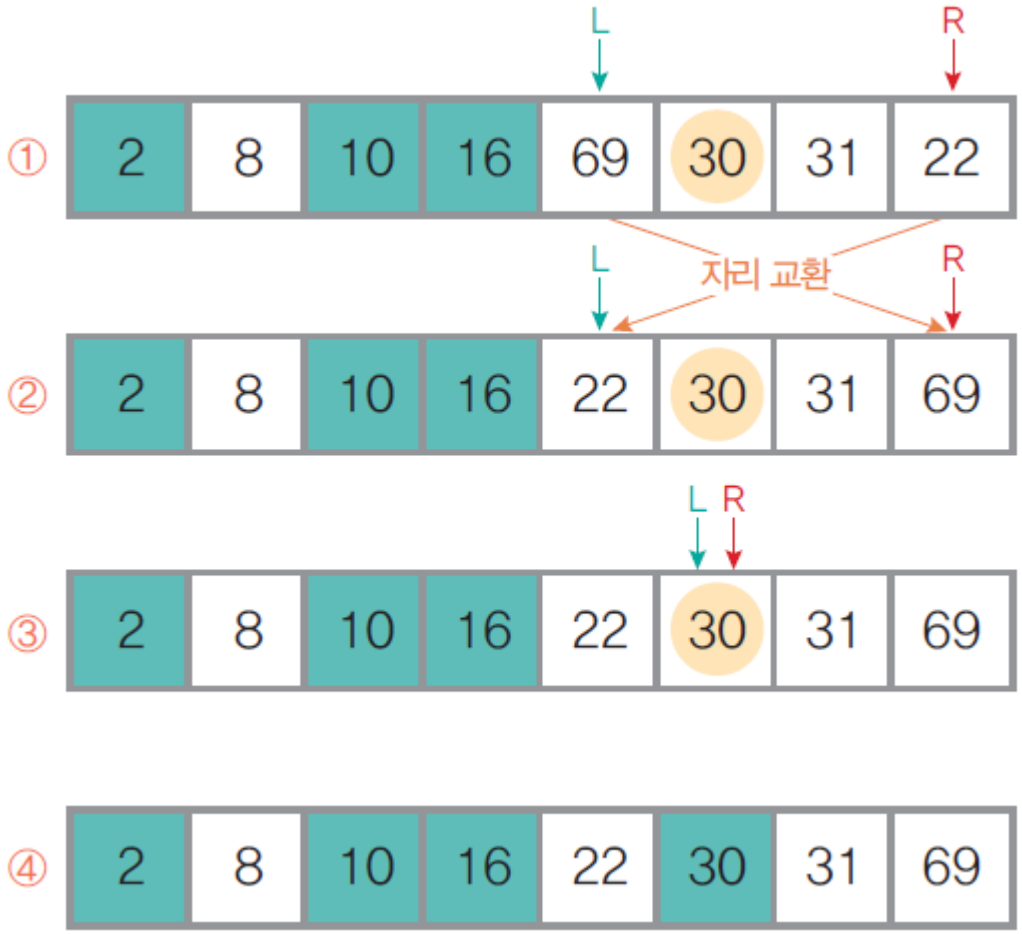
① 오른쪽 부분집합의 원소가 네 개이므로 가운데 있는 원소 30을 피봇으로 선택. L은 오른쪽으로 이동하면서 피봇보다 크거나 같은 원소를 찾고, R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음

② L이 찾은 69와 R이 찾은 22를 서로 교환. 현재 위치에서 다시 L은 피봇보다 크거나 같은 원소를 찾아 오른쪽으로 움직이고 R은 피봇보다 작은 원소를 찾아 왼쪽으로 움직이다가

③ 원소 30에서 L과 R이 만나 멈춤. 더 이상 진행할 수 없으므로

④ R의 원소와 피봇을 교환하고 피봇 원소30의 위치가 확정
(이 경우에 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음).

4. 퀵 정렬


①
2 8 10 16 69 30 31 22
L
R
자리 교환
②
2 8 10 16 22 30 31 69
③
2 8 10 16 22 30 31 69
④
2 8 10 16 22 30 31 69

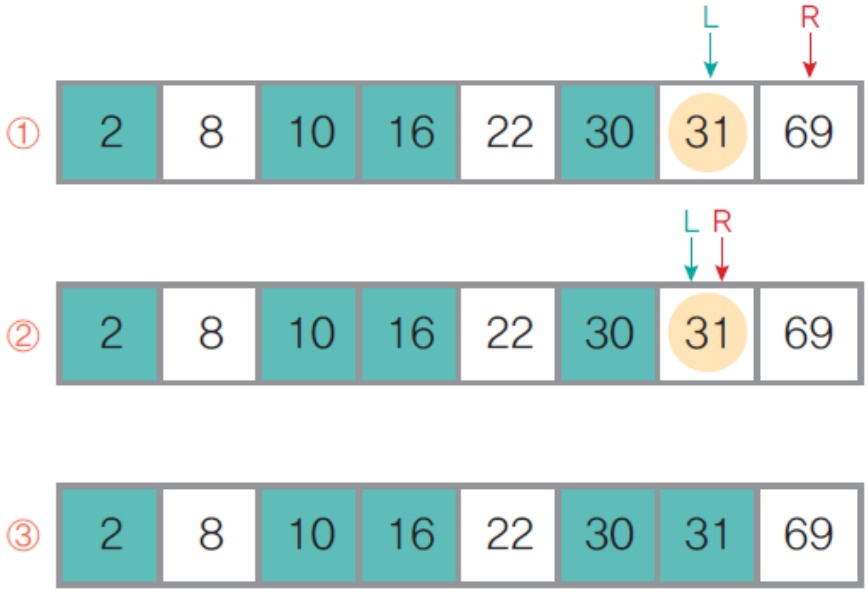
4. 퀵 정렬

(5) 5단계 : 피봇 30의 왼쪽 부분집합의 원소가 한 개이므로 퀵 정렬을 수행하지 않고, 오른쪽 부분집합에 대해서 퀵 정렬을 수행

① 이때 피봇은 31을 선택하고 L은 오른쪽으로 움직이면서 피봇보다 크거나 같은 원소를 찾고 R은 왼쪽으로 움직이면서 피봇보다 작은 원소를 찾음

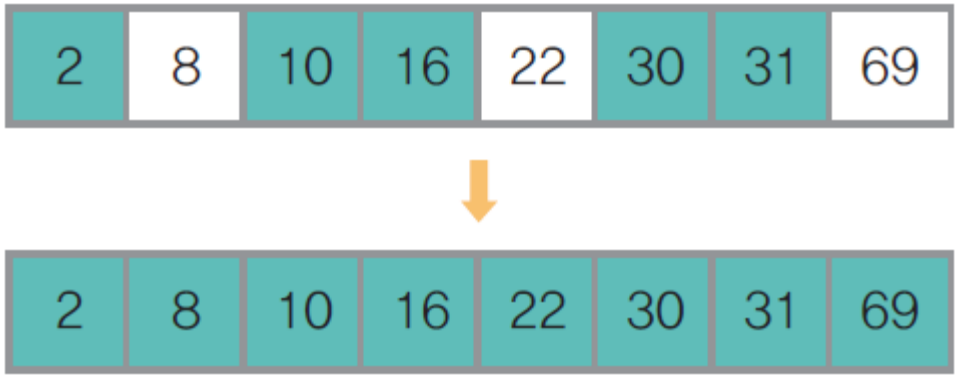
② L과 R이 원소 31에서 만나 더 이상 진행하지 못하는 상태가 되어

③ 원소 31을 피봇과 교환하여 위치를 확정(이 경우에는 R과 피봇 위치가 같았으므로, 자리를 교환하기 전과 후의 상태가 같음).



4. 퀵 정렬

⑹ 6단계 : 피봇 31의 오른쪽 부분집합의 원소가 한 개이므로 퀵 정렬을 수행하지 않지 않음. 모든 부분 집합의 크기가 1 이하이므로 전체 퀵 정렬을 종료



4. 퀵 정렬

❖ 퀵 정렬 알고리즘

- 크기가 n인 배열 a[ ]의 원소를 퀵 정렬하는 알고리즘

알고리즘 9-3 퀵 정렬

```
quickSort(a[], begin, end)
    if (m < n) then {
        p ← partition(a, begin, end);
        quickSort(a[], begin, p-1);
        quickSort(a[], p+1, end);
    }
end quickSort()
```

4. 퀵 정렬

- 퀵 정렬 알고리즘은 배열 a[ ]를 두 개로 분할하는 partition ( ) 연산이 필요

알고리즘 9-4 파티션 분할

```
partiton(a[], begin, end)
    pivot ←(begin + end) / 2;
    L ← begin;
    R ← end;
    while (L<R) do {
        while (a[L]<a[pivot] and L<R) do L++;
        while (a[R]≥a[pivot] and L<R) do R--;
        if (L<R) then {  // L의 원소와 R의 원소 교환
            temp ← a[L];
            a[L] ← a[R];
            a[R] ← temp;
        }
    }
    temp ← a[pivot];    // R의 원소와 피봇 원소 교환
    a[pivot] ← a[R];
    a[R] ← temp;
    return R;           // 교환된 R의 자리를 분할 기준 자리로 반환
end partition()
```

4. 퀵 정렬

- 메모리 사용공간
  - n개의 원소에 대하여 n개의 메모리 사용
- 연산 시간
  - 최선의 경우
    - 피봇에 의해서 원소들이 왼쪽 부분 집합과 오른쪽 부분 집합으로 정확히 n/2개씩 2등분이 되는 경우가 반복되어 수행 단계 수가 최소가 되는 경우
  - 최악의 경우
    - 피봇에 의해 원소들을 분할하였을 때 한개와 n-1개로 한쪽으로 치우쳐 분할되는 경우가 반복되어 수행 단계 수가 최대가 되는 경우
  - 평균 시간 복잡도 : $O(n \log_2 n)$
    - 같은 시간 복잡도를 가지는 다른 정렬 방법에 비해서 자리 교환 횟수를 줄임으로써 더 빨리 실행되어 실행 시간 성능이 좋은 정렬 방법임.

5. 삽입 정렬

❖ 삽입 정렬insert sort의 이해

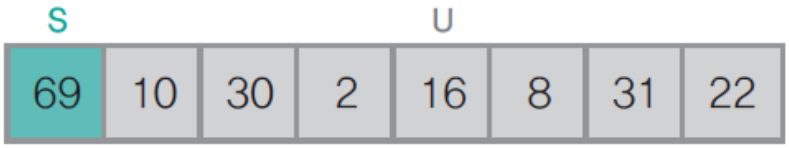
- 정렬되어있는 부분집합에 정렬할 새로운 원소의 위치를 찾아 삽입하는 방법
- 정렬할 자료를 두 개의 부분집합 SSorted Subset와 UUnsorted Subset로 가정
  - 부분집합 S : 정렬된 앞부분의 원소들
  - 부분집합 U : 아직 정렬되지 않은 나머지 원소들
  - 정렬되지 않은 부분집합 U의 원소를 하나씩 꺼내서 이미 정렬되어있는 부분집합 S의 마지막 원소부터 비교하면서 위치를 찾아 삽입
  - 삽입 정렬을 반복하면서 부분집합 S의 원소는 하나씩 늘리고 부분집합 U의 원소는 하나씩 감소하게 함. 부분집합 U가 공집합이 되면 삽입 정렬이 완성

5. 삽입 정렬

❖ 삽입 정렬 수행 과정

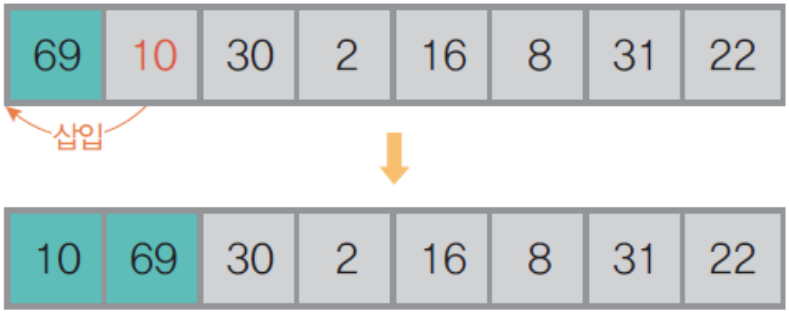
- 정렬하지 않은 {69, 10, 30, 2, 16, 8, 31, 22} 자료를 삽입 정렬 방법으로 정렬하는 과정

0. 초기 상태 : 첫째 원소는 정렬되어 있는 부분집합 S로, 나머지 원소는 정렬되지 않은 원소들의 부분집합 U로 가정



S = {69}
U = {10, 30, 2, 16, 8, 31, 22}

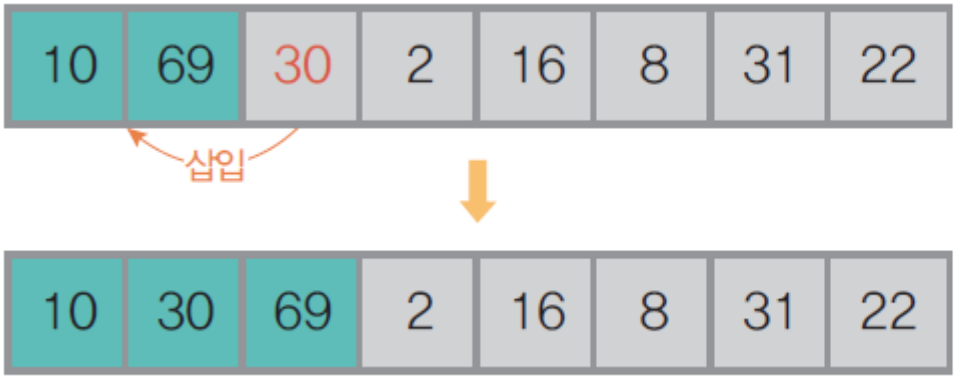
① 1단계 : U의 첫째 원소 10을 S의 마지막 원소 69와 비교하면 10 < 69이므로 원소 10은 원소 69의 앞자리가 됨. 더 이상 비교할 S의 원소가 없으므로 찾은 위치에 원소 10을 삽입



S = {10, 69}
U = {30, 2, 16, 8, 31, 22}

5. 삽입 정렬

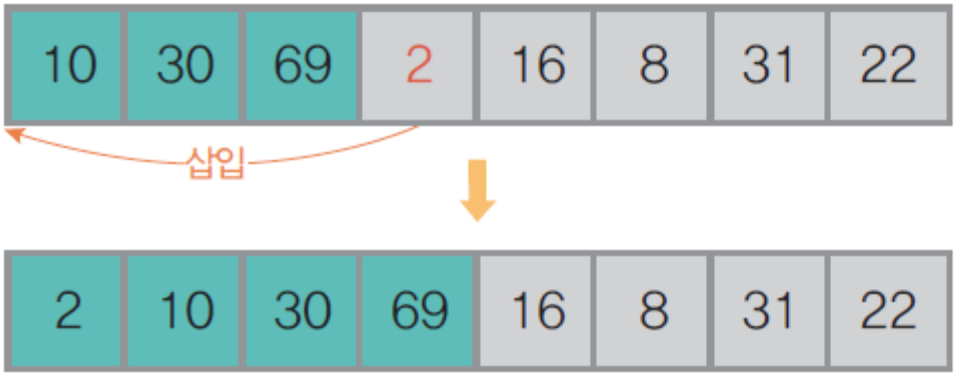
② 2단계 : 현재 U의 첫째 원소 30을 S의 마지막 원소 69와 비교하면 30 < 69이므로 원소 69의 앞자리 원소 10과 비교. 30 > 10이므로 원소 10과 69 사이에 삽입



S = {10, 30, 69}
U = {2, 16, 8, 31, 22}

5. 삽입 정렬

③ 3단계 : 현재 U의 첫째 원소 2를 S의 마지막 원소 69와 비교하면 2 < 69이므로 원소 69의 앞자리 원소30과 비교. 2 < 30이므로 다시 원소 30의 앞자리 원소 10과 비교. 2 < 10이고 더 이상 비교할 S의 원소가 없으므로 원소 10 앞에 삽입



S = {2, 10, 30, 69}
U = {16, 8, 31, 22}

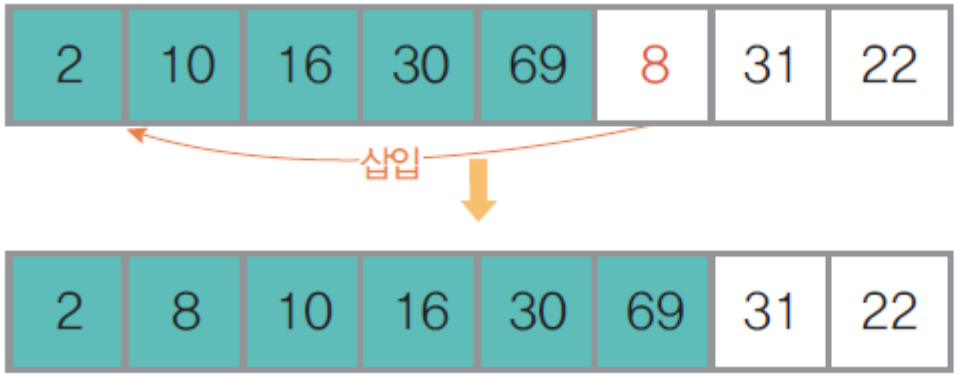
5. 삽입 정렬

④ 4단계 : 현재 U의 첫째 원소 16을 S의 마지막 원소 69와 비교하면 16 < 69 이므로 그 앞자리 원소 30과 비교. 16 < 30이므로 다시 그 앞자리 원소 10과 비교. 16 > 10이므로 원소 10과 30 사이에 삽입

S = {2, 10, 16, 30, 69}
U = {8, 31, 22}

5. 삽입 정렬

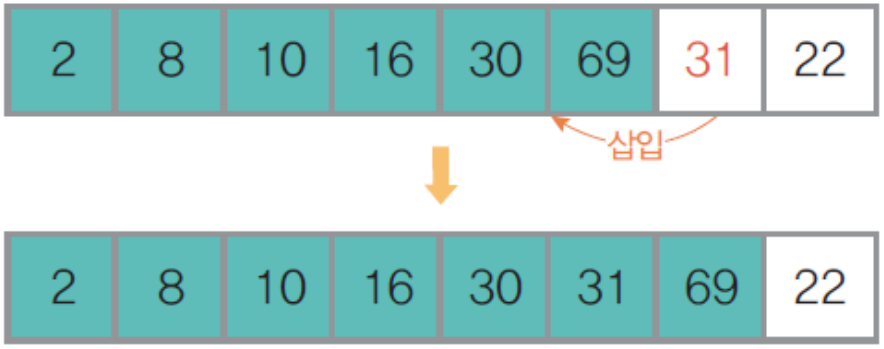
⑤ 5단계 : 현재 U의 첫 번째 원소 8을 S의 마지막 원소 69와 비교하면 8 < 69 이므로 그 앞자리 원소 30과 비교. 8 < 30이므로 그 앞자리 원소 16과 비교. 8 < 16이므로 그 앞자리 원소 10과 비교. 8 < 10이므로 다시 그 앞자리 원소 2와 비교. 8 > 2이므로 원소 2와 10 사이에 삽입



S = {2, 8, 10, 16, 30, 69}
U = {31, 22}

5. 삽입 정렬

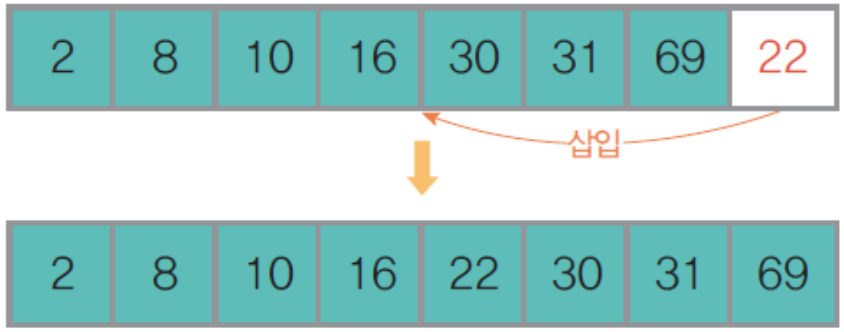
⑥ 6단계 : 현재 U의 첫째 원소 31을 S의 마지막 원소 69와 비교하면 31 < 69 이므로 그 앞자리 원소 30과 비교. 31 > 30이므로 원소 30과 69 사이에 삽입



S = {2, 8, 10, 16, 30, 31, 69}
U = {22}

5. 삽입 정렬

⑦ 7단계 : 현재 U의 첫째 원소 22를 S의 마지막 원소 69와 비교하면 22 < 69 이므로 그 앞자리 원소 31과 비교. 22 < 31이므로 그 앞자리 원소 30과 비교. 22 < 30이므로 다시 그 앞자리 원소 16과 비교. 22 > 16이므로 원소 16과 30 사이에 삽입. U가 공집합이 되었으므로 실행을 종료



S = {2, 8, 10, 16, 22, 30, 31, 69}
U = { }

5. 삽입 정렬

- 크기가 n인 배열 a[ ]의 원소를 삽입 정렬하는 알고리즘

알고리즘 9-5 삽입 정렬

```
insertionSort(a[], n)
    for (i ← 1; i<n; i ← i + 1) do {
        temp ← a[i];
        j ← i;
        if (a[j - 1]>temp) then move ← true;  // 삽입 자리를 만들기 위해 이동할 원소 표시
        else move ← false;
        while (move and j>0) do {             // 삽입 자리를 만들기 위해
            a[j] ← a[j - 1];                  // 삽입 자리 이후의 원소를 뒤로 이동
            j ← j - 1;
        }
        a[j] ← temp;                          // 삽입 자리에 원소 저장
    }
end insertionSort()
```

5. 삽입 정렬

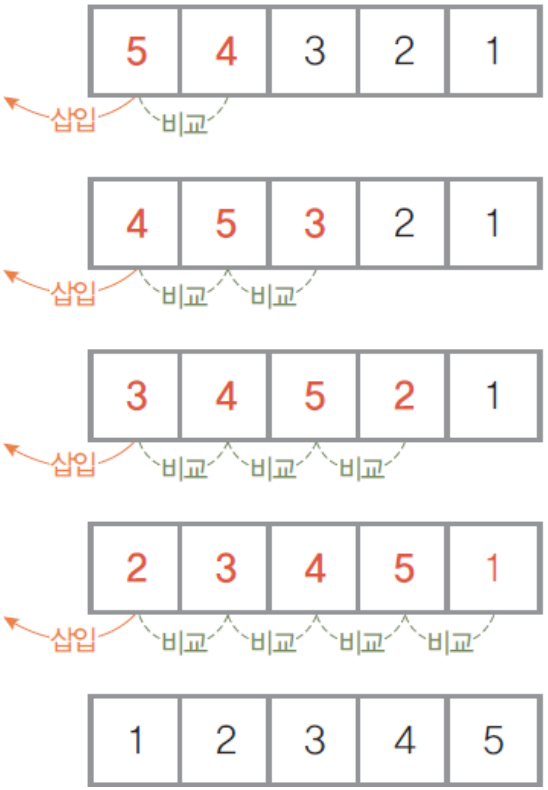
- 메모리 사용공간
  - n개의 원소에 대하여 n개의 메모리 사용
- 연산 시간
  - 최선의 경우 : 원소들이 이미 정렬되어있어서 비교횟수가 최소인 경우
    - 이미 정렬되어있는 경우에는 바로 앞자리 원소와 한번만 비교
    - 전체 비교횟수 = n-1
    - 시간 복잡도 : O(n)
  - 최악의 경우 : 모든 원소가 역순으로 되어있어서 비교횟수가 최대인 경우
    - 전체 비교횟수 = 1+2+3+ … +(n-1) = n(n-1)/2
    - 시간 복잡도 : $O(n^2)$
  - 삽입 정렬의 평균 비교횟수 = n(n-1)/4
  - 평균 시간 복잡도 : $O(n^2)$

5. 삽입 정렬

- 전체 비교 횟수 : n−1
- 전체 자리 이동 횟수 : 0

→ 시간 복잡도 : O(n)

(a) 순차 정렬된 경우



- 전체 비교 횟수 : $1+2+3+\cdots+(n-1) = \sum_{i=1}^{n-1} i$
- 전체 자리 이동 횟수 : $1+2+3+\cdots+(n-1) = \sum_{i=1}^{n-1} i$
- 실행 연산 횟수 : $\sum_{i=1}^{n-1}$(비교 횟수 + 자리 이동 횟수) $= \sum_{i=1}^{n-1} (i + i)$

→ 시간 복잡도 : $O(n^2)$

(b) 역순 정렬된 경우

그림 9-4 순차 정렬된 경우와 역순 정렬된 경우의 삽입 정렬 비교

Thank You