1 쉘 정렬

1 쉘 정렬

1 쉘 정렬(Shell Sort)

- 주어진 입력 데이터를 적당한 매개 변수의 값만큼 서로 떨어진 데이터들과 비교하여 교환하는 과정을 매개 변수의 값을 바꾸어가며 되풀이하는 것
- 영역을 나눈 후 삽입하는 **보완된 삽입 정렬** 개념
- 입력 파일을 어떤 매개변수의 값으로 서브파일을 구성하고, 각 서브파일을 삽입 정렬 방식으로 배열하는 과정을 반복하는 정렬 방식

1 쉘 정렬(Shell Sort)

- 이때 서로 떨어져 있는 데이터들은 하나의 부분 리스트를 구성하며 삽입 정렬에 의해 개별적으로 정렬 됨

- 먼저 적절한 매개 변수의 값을 선택한 후 점차 감소시켜 가면서 쉘 정렬을 수행하고 **매개 변수의 값이 1이 될 때 종료함**

※ 참고: 삽입 정렬은 거의 정렬된 입력에 대해 다른 정렬 알고리즘보다 빠름

1 쉘 정렬

2 쉘 정렬의 작동 예

◆ 예) 다음의 데이터를 쉘 정렬로 정렬하시오.

30	60	90	10	40	80	40	20	10	60	50	30	40	90	80

풀이)

- 먼저 간격(Gap)이 5가 되는 숫자끼리 그룹을 만듦
- 총 15개의 숫자가 있으므로 첫 번째 그룹은 첫 숫자인 30, 첫 숫자에서 간격이 5되는 숫자인 80, 그리고 80에서 간격이 5인 50으로 구성됨

1 쉘 정렬

2 쉘 정렬의 작동 예

◆ 예) 다음의 데이터를 쉘 정렬로 정렬하시오.

30	60	90	10	40	80	40	20	10	60	50	30	40	90	80

풀이)

- 첫 번째 그룹은 [30, 80, 50]임
- 두 번째 그룹은 [60, 40, 30]이고,
  나머지 그룹은 각각 [90, 20, 40], [10, 10, 90],
  [40, 60, 80]임

1 쉘 정렬

2 쉘 정렬의 작동 예

h=5

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
그룹 1	30	60	90	10	40	80	40	20	10	60	50	30	40	90	80
그룹 2															
그룹 3															
그룹 4															
그룹 5															

- 각 그룹 별로 삽입 정렬을 수행한 결과를 나열하면 다음과 같음

30 30 20 10 40 50 40 40 10 60 80 60 90 90 80

1 쉘 정렬

2 쉘 정렬의 작동 예

A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
그룹 1	30	30	20	10	40	50	40	40	10	60	80	60	90	90	80
그룹 2															
그룹 3															
그룹 4															
그룹 5															

그룹별 정렬 후

- 간격이 5인 그룹 별로 정렬한 결과를 보면
  80과 90 같은 큰 숫자가 뒷부분으로 이동하였고,
  20과 30같은 작은 숫자가 앞부분으로
  이동한 것을 알 수 있음

1 쉘 정렬

2 쉘 정렬의 작동 예

- 그 다음엔 간격을 5보다 작게 하여 예를 들어 3으로 하여 3개의 그룹으로 나누어 각 그룹별로 삽입 정렬을 수행하는데 이때에는 각 그룹에 5개의 숫자가 있음
- 마지막에는 **반드시 간격을 1**로 놓고 수행해야 하는데 다른 그룹에 속해 서로 비교되지 않은 숫자가 있을 수 있기 때문임
- 즉, 모든 원소를 1개의 그룹으로 여기는 것이고 이는 삽입 정렬임

1 쉘 정렬

2 쉘 정렬의 작동 예

- 간격을 3으로 하고 정렬한 결과

10 30 10 30 40 20 40 40 50 60 80 60 90 90 80

- 간격을 2로 하고 정렬한 결과

10 20 10 30 40 30 40 40 50 60 80 60 80 90 90

- 간격을 1로 하고 정렬한 결과(정렬 완료)

10 10 20 30 30 40 40 40 50 60 60 80 80 90 90

1 쉘 정렬

3 쉘 정렬 복잡도

- 쉘 정렬은 최악 경우(Worst Case)의 시간복잡도가 $O(n^2)$임
- 쉘 정렬의 수행 속도는 **간격 선정에 따라 좌우**됨

1 쉘 정렬

4 쉘 정렬의 특징

- 쉘 정렬은 입력 크기가 매우 크지 않은 경우에 매우 좋은 성능을 보임

- 쉘 정렬은 임베디드 시스템에서 주로 사용되는데 쉘 정렬의 특징인 **간격에 따른 그룹 별 정렬 방식**이 하드웨어로 정렬 알고리즘을 구현하는데 매우 적합하기 때문

2 병합 정렬

2 병합 정렬

1 병합 정렬(Merge Sort)

- 정렬할 데이터들을 2개로 나누고 2개로 나뉜 데이터들을 각각 정렬한 다음에 다시 합병하여 하나의 정렬된 데이터들로 완성하는 방법

- 이미 정렬되어 있는 두 개의 파일을 1개의 파일로 합병하는 정렬 방식으로 1개의 파일에서는 각각의 데이터를 하나의 파일로 취급하여 정렬함

2 병합 정렬

1 병합 정렬(Merge Sort)

```
mergeSort(A[ ], p, r)          ----------▷ A[p ... r]을 정렬
{
      if (p < r) then {
            q ← ⌊(p + r)/2⌋;  ----------  ①  ▷ p, r의 중간 지점 계산
            mergeSort(A, p, q);  -------  ②  ▷ 전반부 정렬
            mergeSort(A, q+1, r); -----  ③  ▷ 후반부 정렬
            merge(A, p, q, r);  ---------  ④  ▷ 병합
      }
}

merge(A[ ], p, q, r)
{
      정렬되어 있는 두 배열 A[p ... q]와 A[q+1 ... r]을 합쳐
      정렬된 하나의 배열 A[p ... r]을 만든다.
}
```

2 병합 정렬의 작동 예

정렬할 배열이 주어짐

31	3	65	73	8	11	20	29	48	15

배열을 반반으로 나눈다

31	3	65	73	8	11	20	29	48	15

— ①

각각 독립적으로 정렬한다

3	8	31	65	73	11	15	20	29	48

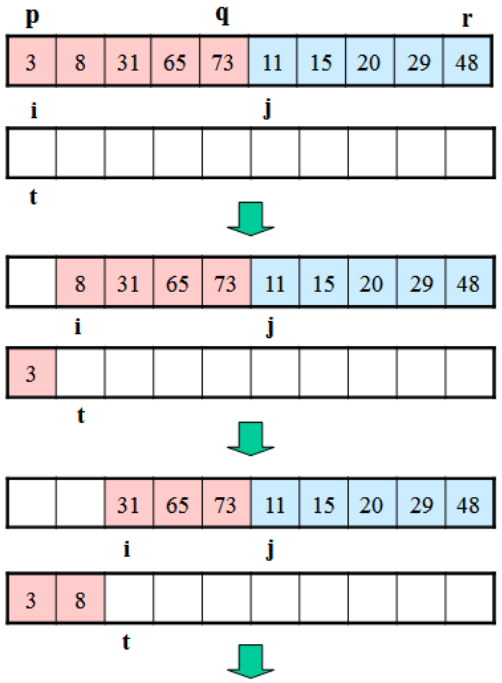
— ② ③

병합한다 (정렬완료)

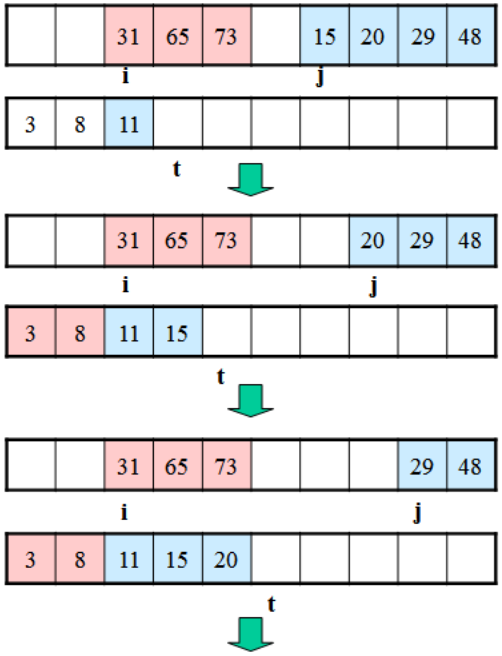
3	8	11	15	20	29	31	48	65	73

— ④

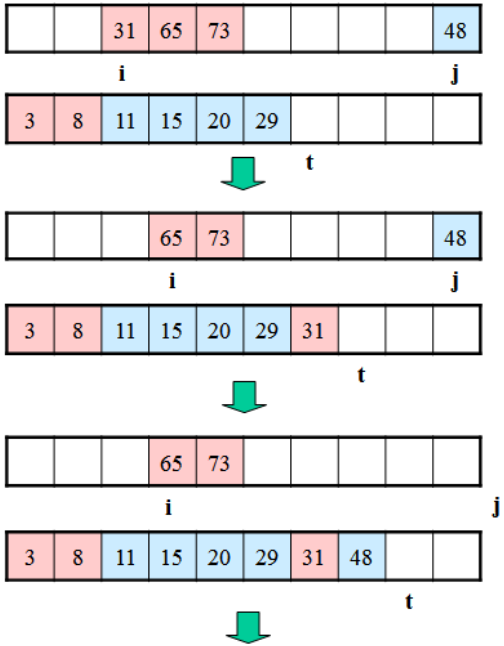
2 병합 정렬의 작동 예


p
q
r
3 8 31 65 73 11 15 20 29 48
i
j
t
8 31 65 73 11 15 20 29 48
i
j
3
t
31 65 73 11 15 20 29 48
i
j
3 8
t

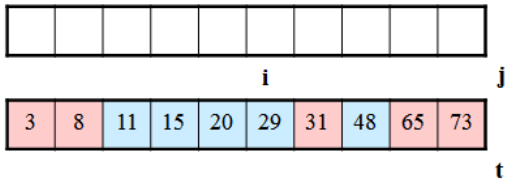
2 병합 정렬의 작동 예


31 65 73 15 20 29 48
i
j
3 8 11
t
31 65 73 20 29 48
i
j
3 8 11 15
t
31 65 73 29 48
i
j
3 8 11 15 20
t

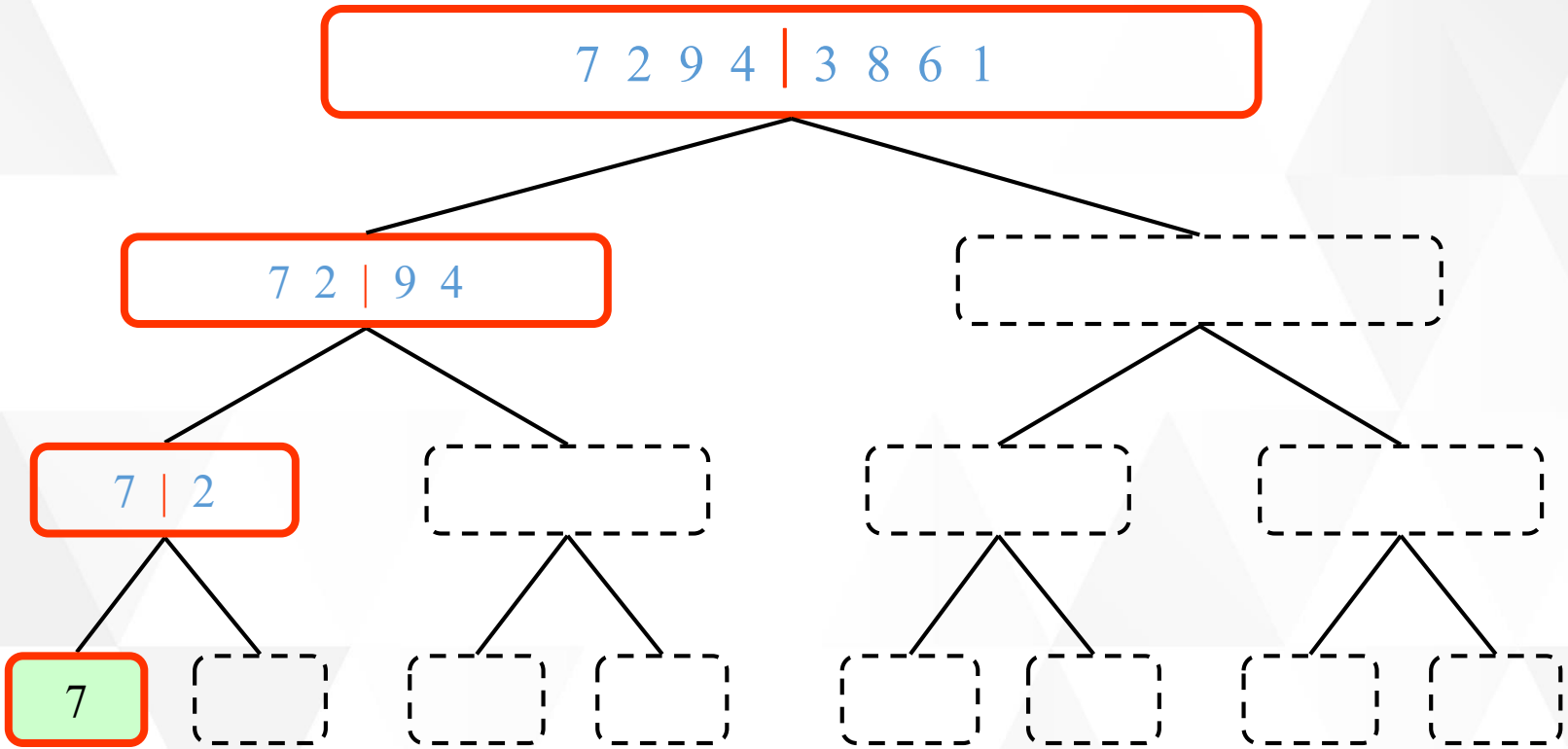
2 병합 정렬의 작동 예


31 65 73 48
i
j
3 8 11 15 20 29
t
65 73 48
i
j
3 8 11 15 20 29 31
t
65 73
i
j
3 8 11 15 20 29 31 48
t

2 병합 정렬의 작동 예

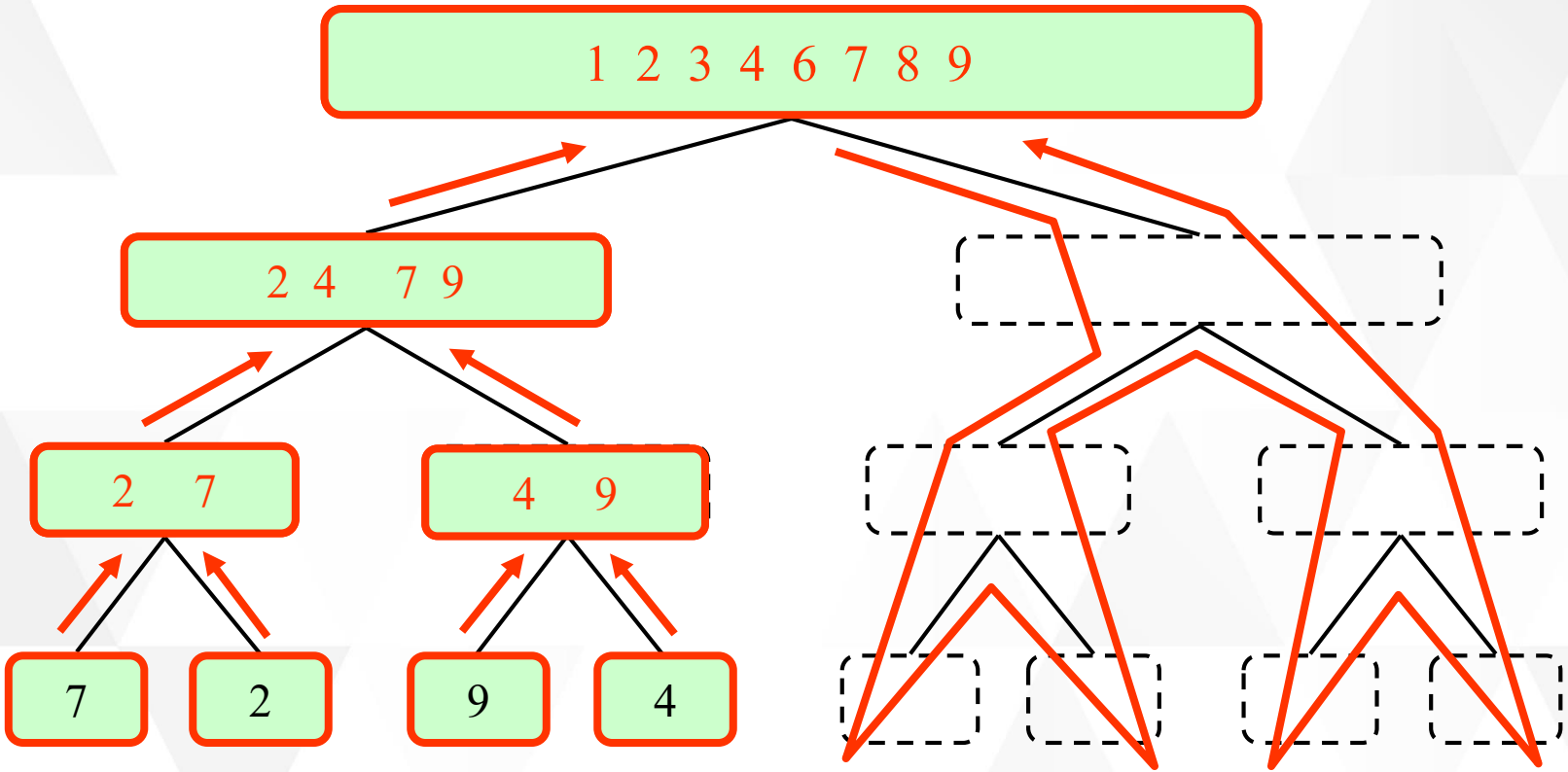


2 병합 정렬의 작동 예



2 병합 정렬의 작동 예

- 수행 시간: O(**n** log**n**)



3 퀵 정렬

3 퀵 정렬

1 퀵 정렬(Quick Sort)

- 기준 키를 기준으로 작거나 같은 값을 지닌 데이터는 앞으로, 큰 값을 지닌 데이터는 뒤로 가도록 하여 작은 값을 갖는 데이터와 큰 값을 갖는 데이터로 분리해가며 정렬하는 방법

- 평균적으로 가장 좋은 성능을 가져 현장에서 가장 많이 쓰이는 정렬 알고리즘

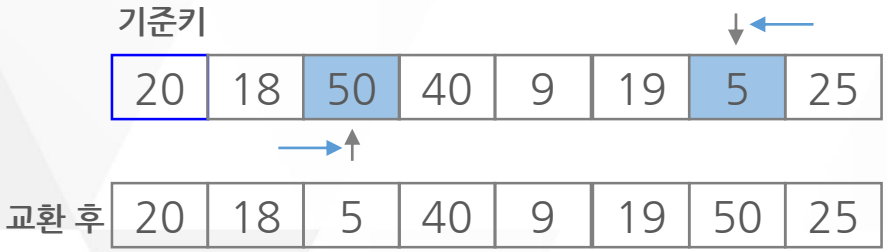
3 퀵 정렬

2 퀵 정렬의 작동 예

[정렬하고자 하는 데이터]

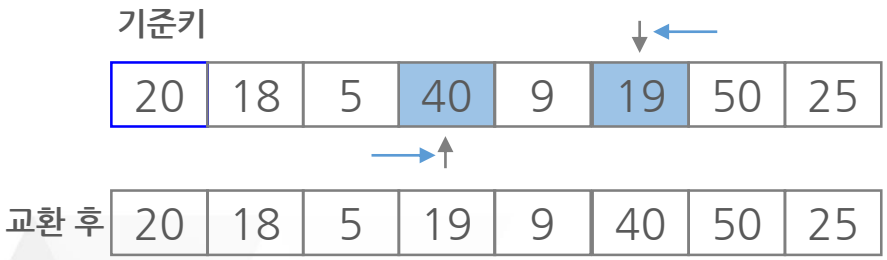
20	18	50	40	9	19	5	25

① 맨 앞의 20을 기준키로 하고 기준키 다음부터 기준키보다 큰 데이터를 찾아 50을 선택하고, 마지막 데이터부터 기준키보다 작은 데이터를 찾아 5를 선택함
그리고 선택된 50과 5를 교환함



2 퀵 정렬의 작동 예

② 계속해서 진행하여 기준키보다 큰 데이터인 40을 선택하고, 기준키보다 작은 데이터인 19를 선택한 후 두 수를 교환함



3 퀵 정렬

2 퀵 정렬의 작동 예

③ 마찬가지로 진행하여 기준키보다 큰 데이터인
40과 기준키보다 작은 데이터인 9를 선택함

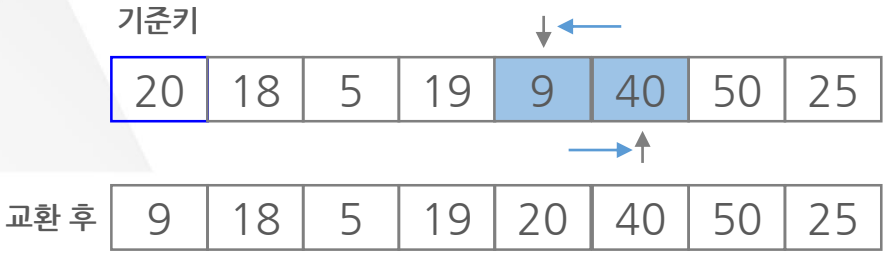
그런데 발견된 위치가 서로 교차하는데
이런 경우에는 두 값을 교환하지 않고 기준키 20과

작은 데이터인 9를 교환함

또한 기준키보다 큰 데이터를 발견하지 못하는
경우에도 기준키와 작은 데이터를 교환함

3 퀵 정렬

2 퀵 정렬의 작동 예



3 퀵 정렬

2 퀵 정렬의 작동 예

④ 데이터들을 보면 기준키 20을 기준으로 왼쪽에는 기준키보다 작은 데이터들이, 오른쪽에는 큰 데이터들이 있음

이때 기준키를 중심으로 양분함

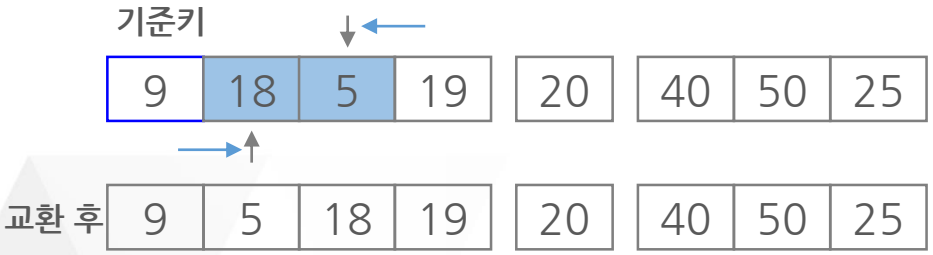
9	18	5	19	20	40	50	25

이제부터는 기준키를 중심으로 왼쪽 데이터들에 대해 그리고 오른쪽 데이터들에 대해 같은 방법으로 동작함

2 퀵 정렬의 작동 예

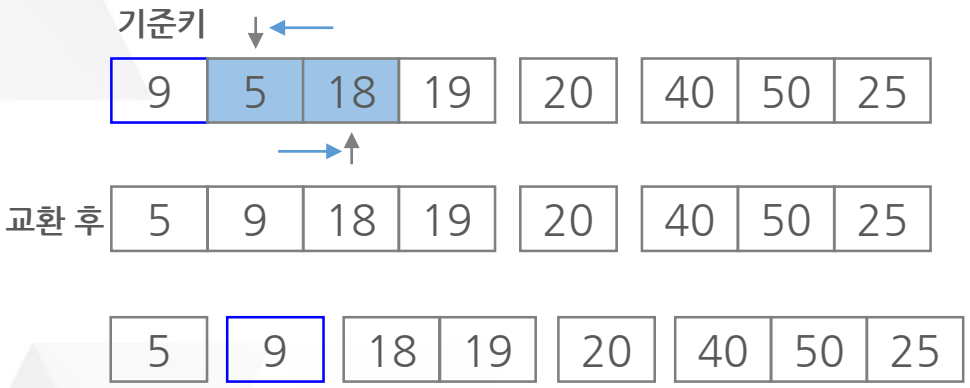
[왼쪽 데이터들에 대해 동작하는 과정]

⑤ 기준키 9보다 큰 데이터인 18과
작은 데이터인 5를 선택하고 교환함



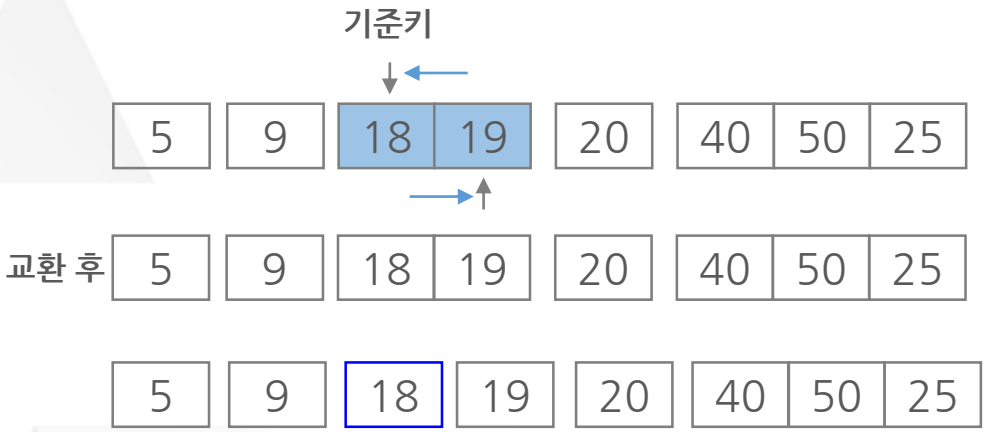
2 퀵 정렬의 작동 예

⑥ 나머지도 마찬가지로 진행하여 정렬함



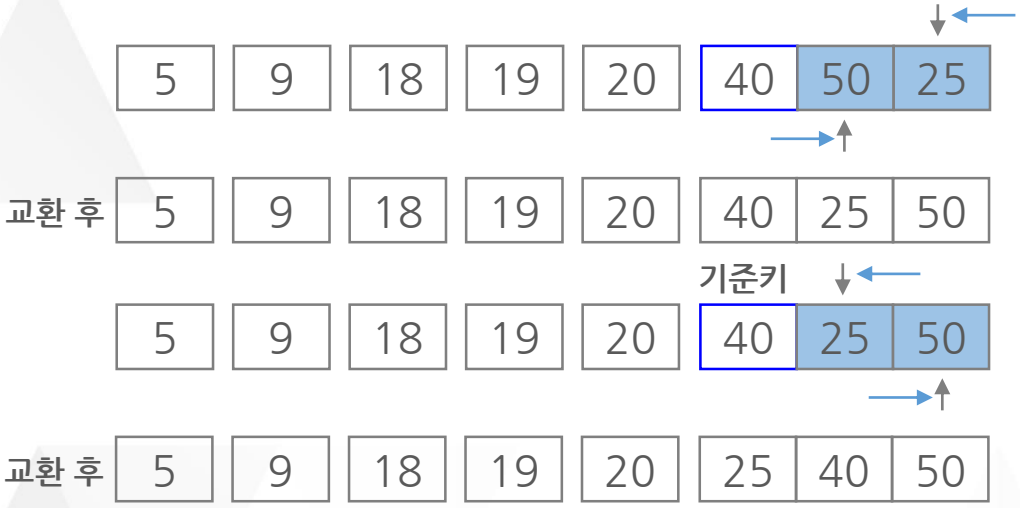
3 퀵 정렬

2 퀵 정렬의 작동 예



3 퀵 정렬

2 퀵 정렬의 작동 예



3 퀵 정렬

2 퀵 정렬의 작동 예

최종 정렬 결과

5	9	18	19	20	25	40	50

3 퀵 정렬

3 퀵 정렬 알고리즘

```
quickSort(A[], p, r)      ▷ A[p ... r]을 정렬한다
{
      if (p < r) then {
            q = partition(A, p, r);  ▷ 분할
            quickSort(A, p, q-1);    ▷ 왼쪽 부분 배열 정렬
            quickSort(A, q+1, r);    ▷ 오른쪽 부분 배열 정렬
      }
}


partition(A[], p, r)
{
      배열 A[p ... r]의 원소들을 A[r]을 기준으로 양쪽으로 재배치하고
      A[r]이 자리한 위치를 리턴한다;
}
```

3 퀵 정렬

4 퀵 정렬 복잡도

- 평균 수행 시간(Average Case) : O(n logn)
- 최악 수행 시간(Worst Case) : $O(n^2)$