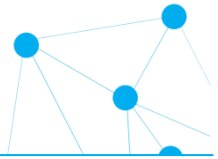
데이터구조 5장

05-1. 정렬 개념

정렬이란?

- 물건을 오름/내림차순으로 나열하는 것
  - 정렬은 컴퓨터공학을 포함한 모든 과학기술 분야에서 가장 기본적이고 중요한 알고리즘 중 하나

정렬 알고리즘 개요



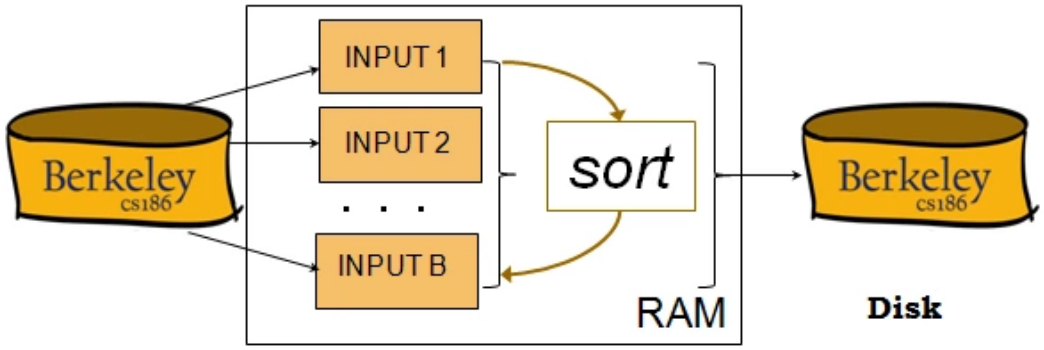
- 모든 경우에 대해 최적인 정렬 알고리즘은 없음

  ➔ 해당 응용 분야에 적합한 정렬 방법 사용해야 함

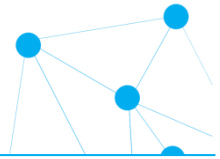
  – 레코드 수의 많고 적음 / 레코드 크기의 크고 작음

  – Key의 특성(문자, 정수, 실수 등)

  – 메모리 내부/외부 정렬



정렬 알고리즘 분류

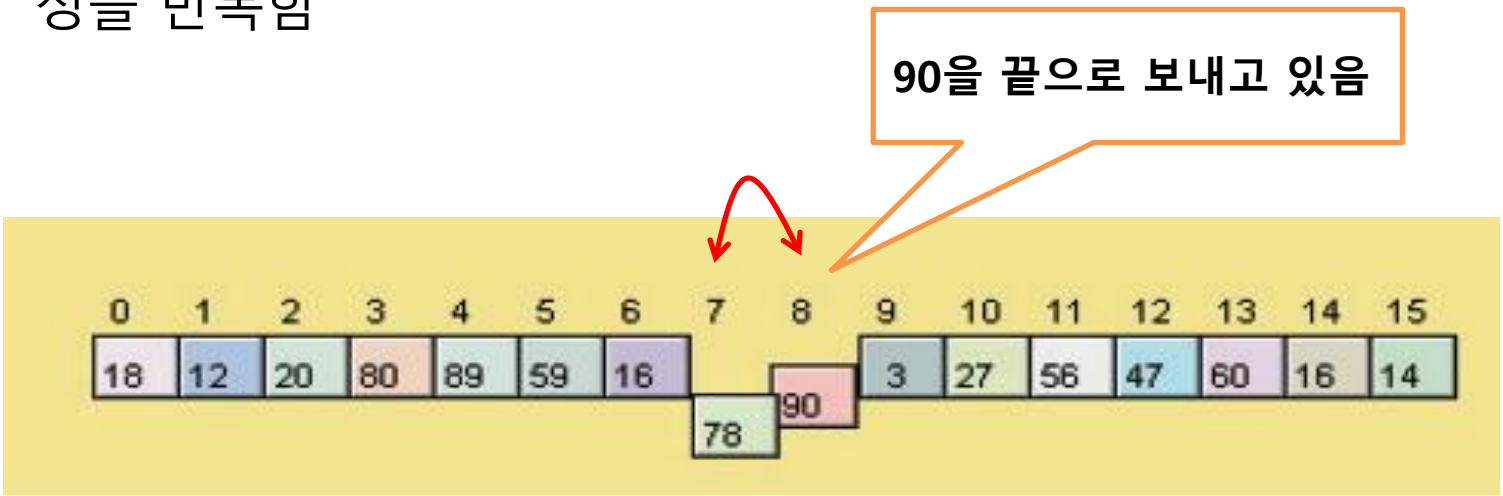


- 복잡도와 효율성에 따른 분류
  - 단순하지만 비효율적인 방법 : 삽입, 선택, 버블정렬 등
  - 복잡하지만 효율적인 방법 : 퀵, 힙, 합병, 기수정렬 등
- 정렬장소에 따른 분류
  - 내부 정렬(internal sorting) : 모든 데이터가 주기억장치에 저장되어진 상태에서 정렬
  - 외부 정렬(external sorting) : 외부 기억장치에 대부분의 데이터가 있고 일부만 주기억장치에 저장된 상태에서 정렬

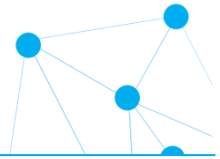
05-2. 버블과 선택

버블정렬 (bubble sort)

- 인접한 2개의 레코드를 비교하여 순서대로 되어 있지 않으면 서로 교환하여 정렬해 나가는 방법
  - 이러한 비교-교환 과정을 리스트의 왼쪽 끝에서 오른쪽 끝까지 반복(스캔)
  - 한번의 스캔이 완료되면 리스트의 오른쪽 끝에 가장 큰 레코드가 이동함
  - 끝으로 이동한 레코드를 제외한 왼쪽 리스트에 대하여 스캔 과정을 반복함



버블정렬 알고리즘



```
Bubble_Sort(A, n) :

for i←n-1 to 1
   for j←0 to i-1
          j와 j+1번째의 요소가 크기 순이 아니면 교환
          j++;
       i--;
```

데이터가 역순으로 정렬되어 있는 경우 많은 이동이 일어남

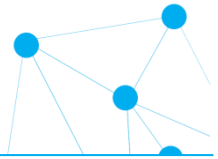
단순하지만 과다한 자료이동으로 인해 잘 사용되지 않음

비교연산보다 이동연산의 시간이 더 걸림

버블정렬 진행과정

스캔1 정렬과정

선택정렬(selection sort)



- 전체 키들 중에서 가장 작은 값(오름차순 정렬기준)을 반복적으로 선택해 앞쪽으로 이동하는 정렬 방법
- 정렬된 왼쪽 리스트와 정렬 안된 오른쪽 리스트
  - 초기에는 왼쪽 리스트는 비어 있고, 정렬할 숫자들은 모두 오른쪽 리스트에 존재

왼쪽 리스트	오른쪽 리스트	설명
( )	(5,3,8,1,2,7)	초기상태
(1)	(5,3,8,2,7)	1선택
(1,2)	(5,3,8,7)	2선택
(1,2,3)	(5,8,7)	3선택
(1,2,3,5)	(8,7)	5선택
(1,2,3,5,7)	(8)	7선택
(1,2,3,5,7,8)	()	8선택

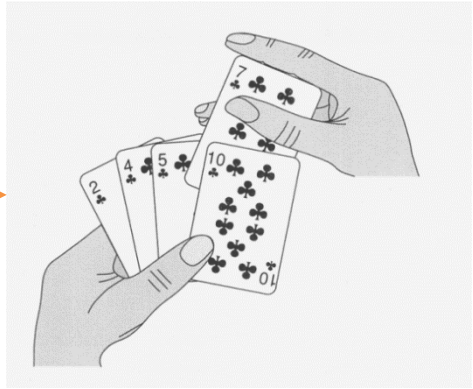
선택정렬 알고리즘


step1
5 3 8 1 2 7
5과 1을 교환
step2
1 3 8 5 2 7
3과 2을 교환
step3
1 2 8 5 3 7
8과 3을 교환
step4
1 2 3 5 8 7
이미제자리에 있음
step5
1 2 3 5 8 7
8과 7을 교환
1 2 3 5 7 8
정렬 완료
1 2 3 5 7 8

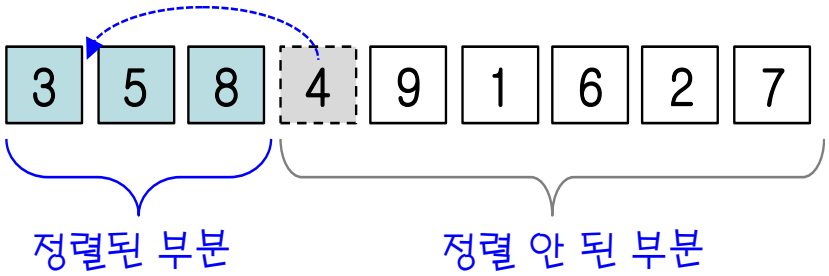
05-3. 삽입과 셸

삽입정렬(insertion sort)

- 이미 정렬되어 있는 부분에 정렬할 키를 적절한 위치로 삽입하는 과정을 반복하는 정렬 방법
  - 카드를 순서대로 정렬하는 것과 유사

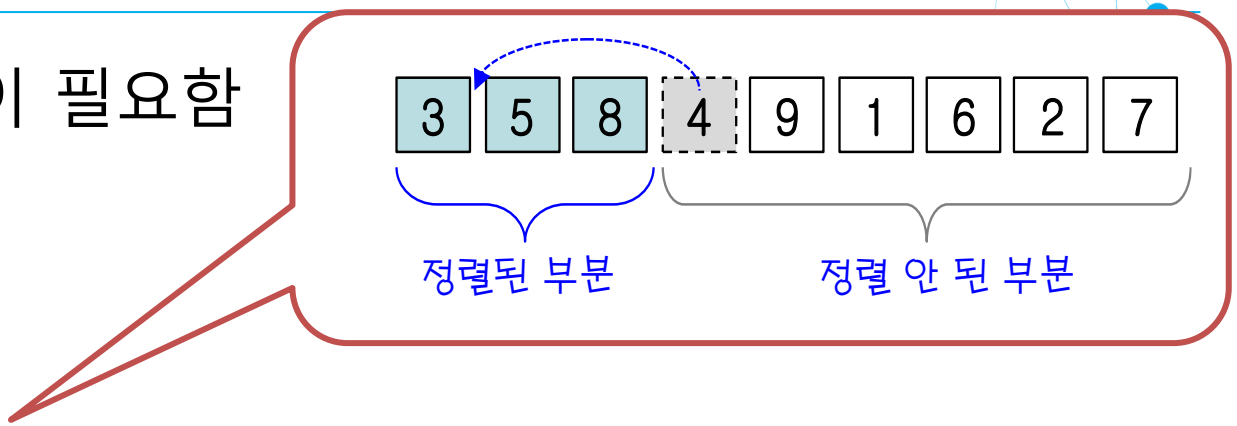


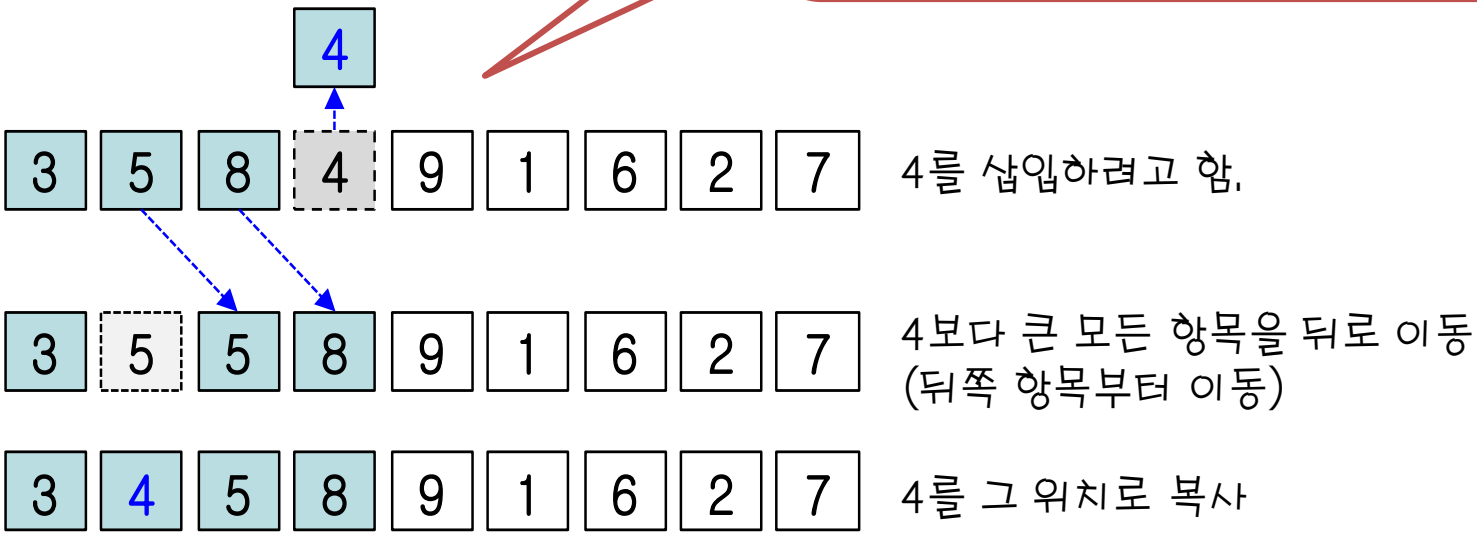
- 정렬된 부분과 정렬 안 된 부분



한번의 삽입 과정

- 항목들의 이동이 필요함





많은 이동 필요 → 레코드가 큰 경우 불리
대부분 데이터가 정렬되어 있다면 매우 효율적인 방법

삽입정렬 알고리즘

인덱스 0은 이미 정렬된 상태로 여김
i는 1에서 출발

```
Insertion_Sort(A, n) :
for i←1 to n-1
    key←A[i];
    j←i-1;
    while j≥0 and A[j]>key
        A[j+1]←A[j];
        j←j-1;
    A[j+1]←key
```

키를 해당 위치에 삽입하기 위해 키보다 앞에 있는 큰 값들을 이동한다.

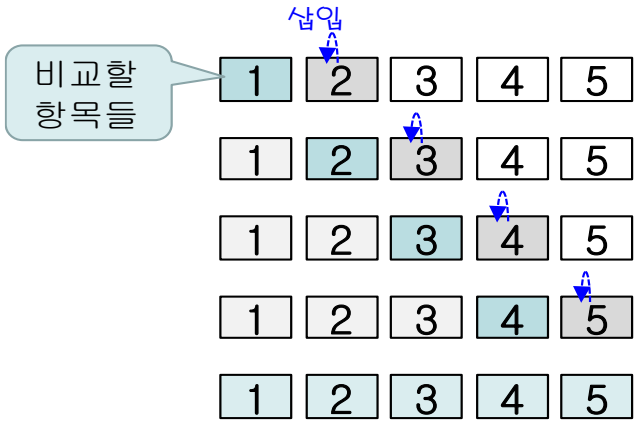
삽입정렬 알고리즘(i=3인경우에)

A[3]=1이 key가 됨
key=A[3]

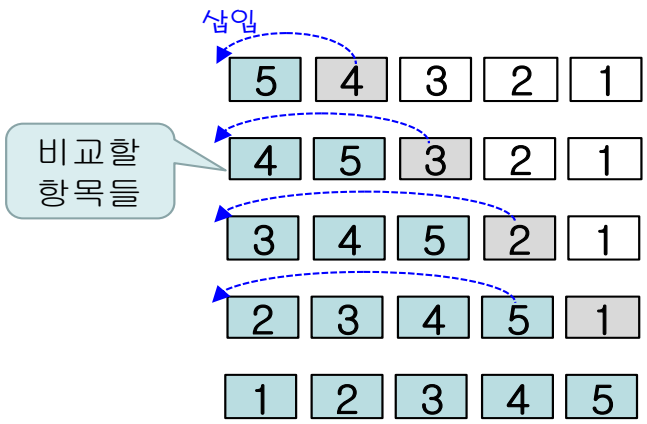
Key와 하나씩 인덱스 앞으로 비교해 가면서 key보다 큰 값들을 뒤로 한칸씩 미룸

셸정렬 (Shell sort)

- 삽입 정렬은 어느 정도 정렬된 상태에서 대단히 빠르다!
  - 그러나 요소들이 이웃한 위치로만 이동➔많은 이동 발생
- 셸 정렬은 전체 리스트를 한꺼번에 정렬하지 않고 일정 기준으로 여러 개의 부분 리스트로 나눈 후 각 리스트를 삽입 정렬하는 방식으로 정렬



(a) 정렬된 배열의 삽입 정렬

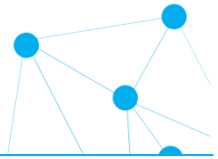


(b)역순 정렬 배열의 삽입 정렬

정렬되어 있어 삽입 정렬시 데이터 이동이 일어나지 않음

정렬되어 있지 않아 삽입 정렬시 많은 데이터 이동 발생

셸정렬 진행과정



1) 리스트를 일정 간격(gap)의 부분 리스트로 나눔
   - 나뉘어진 각각의 부분 리스트를 **삽입정렬** 함
2) 간격을 줄임
   - 부분 리스트의 수는 더 작아지고, 각 부분 리스트는 더 커짐
3) 간격이 1이 될 때까지 이 과정 반복

입력 배열	5	3	8	4	9	1	6	2	7

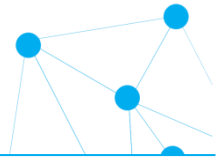
- 간격 k=5의 부분 리스트 정렬 전: {5,1} {3,6}, {8,2}, {4,7}, {9}
- 간격 k=5의 부분 리스트 정렬 후: {1,5} {3,6}, {2,8}, {4,7}, {9}
- Step1 완료: 1, 3, 2, 4, 9, 5, 6, 8, 7

- 간격 k=3의 부분 리스트 정렬 전: {1,4,6} {3,9,8}, {2,5,7}
- 간격 k=3의 부분 리스트 정렬 후: {1,4,6} {3,8,9}, {2,5,7}
- Step2 완료: 1, 3, 2, 4, 8, 5, 6, 9, 7

- 간격 k=1의 부분 리스트 정렬 전: {1,3,2,4,8,5,6,9,7}
- 간격 k=1의 부분 리스트 정렬 후: {1,2,3,4,5,6,7,8,9}
- Step3 완료: 1, 2, 3, 4, 5, 6, 7, 8, 9

마지막엔 전체 데이터를 삽입정렬해야 하지만 거의 정렬된 상태로 출발하므로 효율적으로 삽입정렬 수행

셸정렬 진행과정



- 간격 선택 하기
  - gap은 전체 크기를 반으로 나눈 값에서 출발하고, 해당 값이 짝수이면 하나 큰 값으로 진행하는 것이 효율적임이 증명됨
  - 즉, 간격은 항상 홀수가 됨
- 해당 간격 기준 부분리스트로 나누기
- 각 부분 리스트 삽입정렬하기
- 간격이 1이 될 때가지 과정 반복

입력 배열	**5**	**3**	**8**	**4**	**9**	**1**	**6**	**2**	**7**
간격 5일 때의 부분 리스트	5					1			
		3					6		
			8					2	
				4					7
					9				
Step1 완료	**1**	**3**	**2**	**4**	**9**	**5**	**6**	**8**	**7**
간격 3일 때의 부분 리스트	1			4			6		
		3			9			8	
			2			5			7
Step2 완료	**1**	**3**	**2**	**4**	**8**	**5**	**6**	**9**	**7**
간격 1일 때의 부분 리스트	1	3	2	4	8	5	6	9	7
간격 1 정렬: 1번 비교(1)	1	**3**							
2를 3과 1 각각, 2번 비교(3,1)	1	3	**2**						
4와 3을 1번 비교(3)			3	**4**					
4와 8을 1번 비교(4)				4	**8**				
5를 8과 4에 각각, 2번 비교(8,4)				4	8	**5**			
6을 8, 5에 각각, 2번 비교(8,5)					5	8	**6**		
9를 8과 1번 비교(8)							8	**9**	
7을 9, 8, 6순으로 3번 비교(9,8,7)						6	8	9	**7**
Step3 완료	**1**	**2**	**3**	**4**	**5**	**6**	**8**	**7**	**9**

셸정렬 진행과정

(a) 간격 5로 만들어진 부분 리스트

(b) 간격 5로 만들어진 부분 리스트

리스트1

[10, 3, 16]

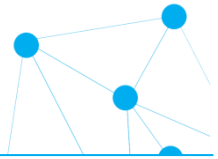
➔ 삽입정렬 [3, 10, 16]

간격 5의 부분 정렬 결과

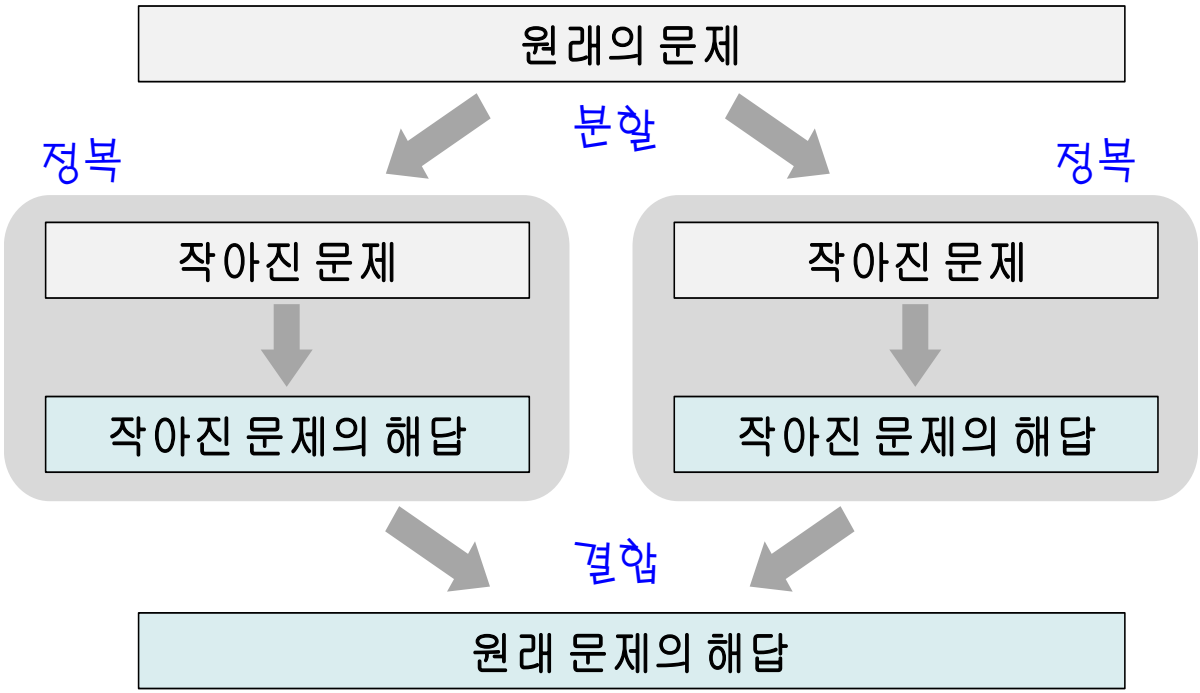
입력 배열	10	8	6	20	4	3	22	1	0	15	16
간격 5일 때의 부분 리스트	10					3					16
		8					22				
			6					1			
				20					0		
					4					15	
부분 리스트 정렬 후	3					10					16
		8					22				
			1					6			
				0					20		
					4					15	
간격 5 정렬후의 전체 배열	3	8	1	0	4	10	22	6	20	15	16
간격 3일 때의 부분 리스트	3			0			22			15	
		8			4			6			16
			1			10			20		
부분 리스트 정렬 후	0			3			15			22	
		4			6			8			16
			1			10			20		
간격 3 정렬 후의 전체 배열	0	4	1	3	6	10	15	8	20	22	16
간격 1 정렬 후의 전체 배열	0	1	3	4	6	8	10	15	16	20	22

05-4. 합병과 퀵, 기수

합병정렬 (Merge Sort)



- 분할 정복(divide and conquer) 방법
  - 문제를 보다 작은 2개의 문제로 분리하고 각 문제를 해결한 다음, 결과를 모아서 원래의 문제를 해결하는 전략

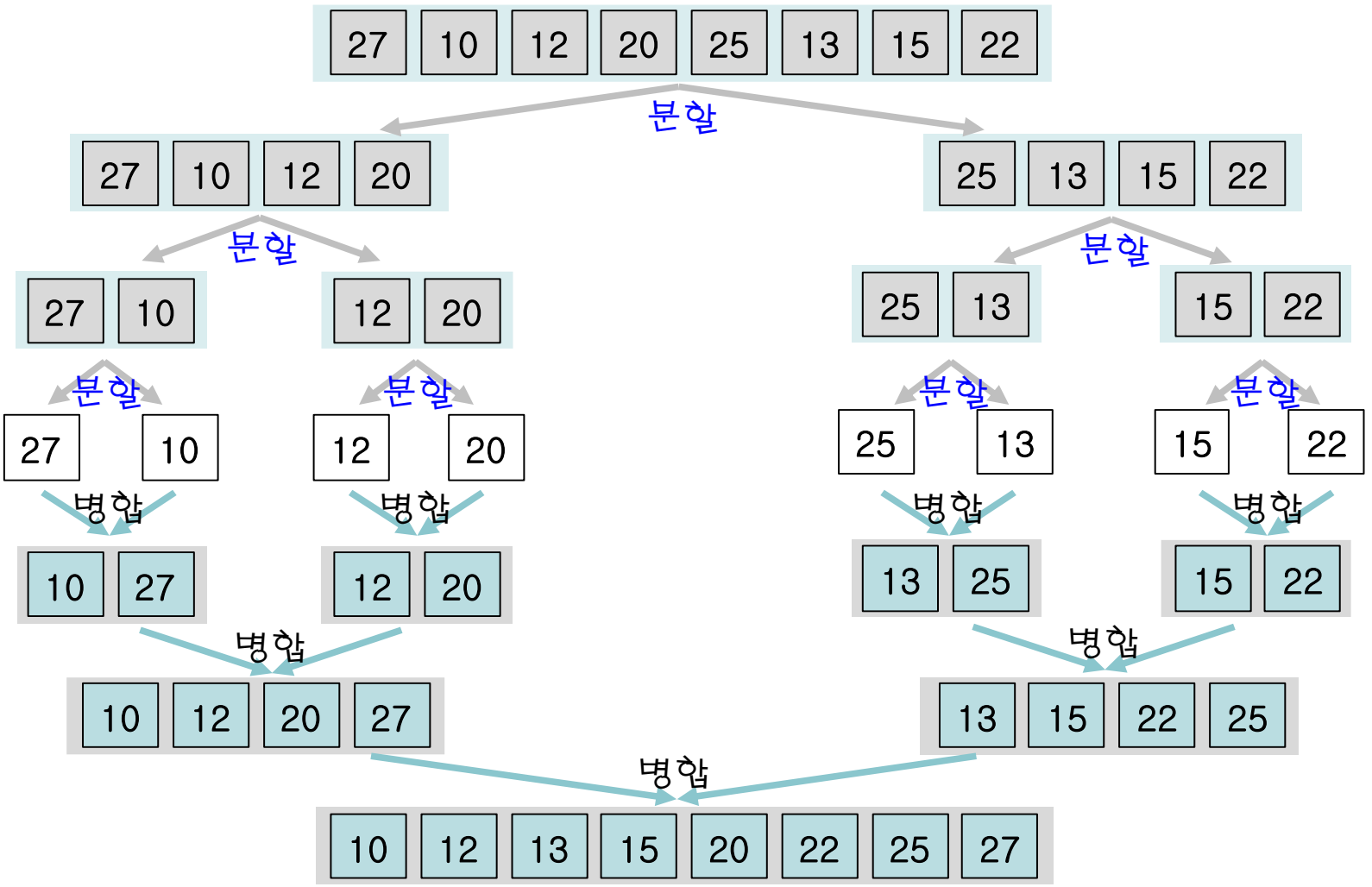


합병정렬

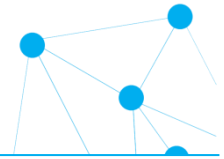
- 리스트를 두 개의 부분 리스트로 분할하고 각각을 정렬
- 정렬된 부분 리스트를 합해 전체 리스트를 정렬

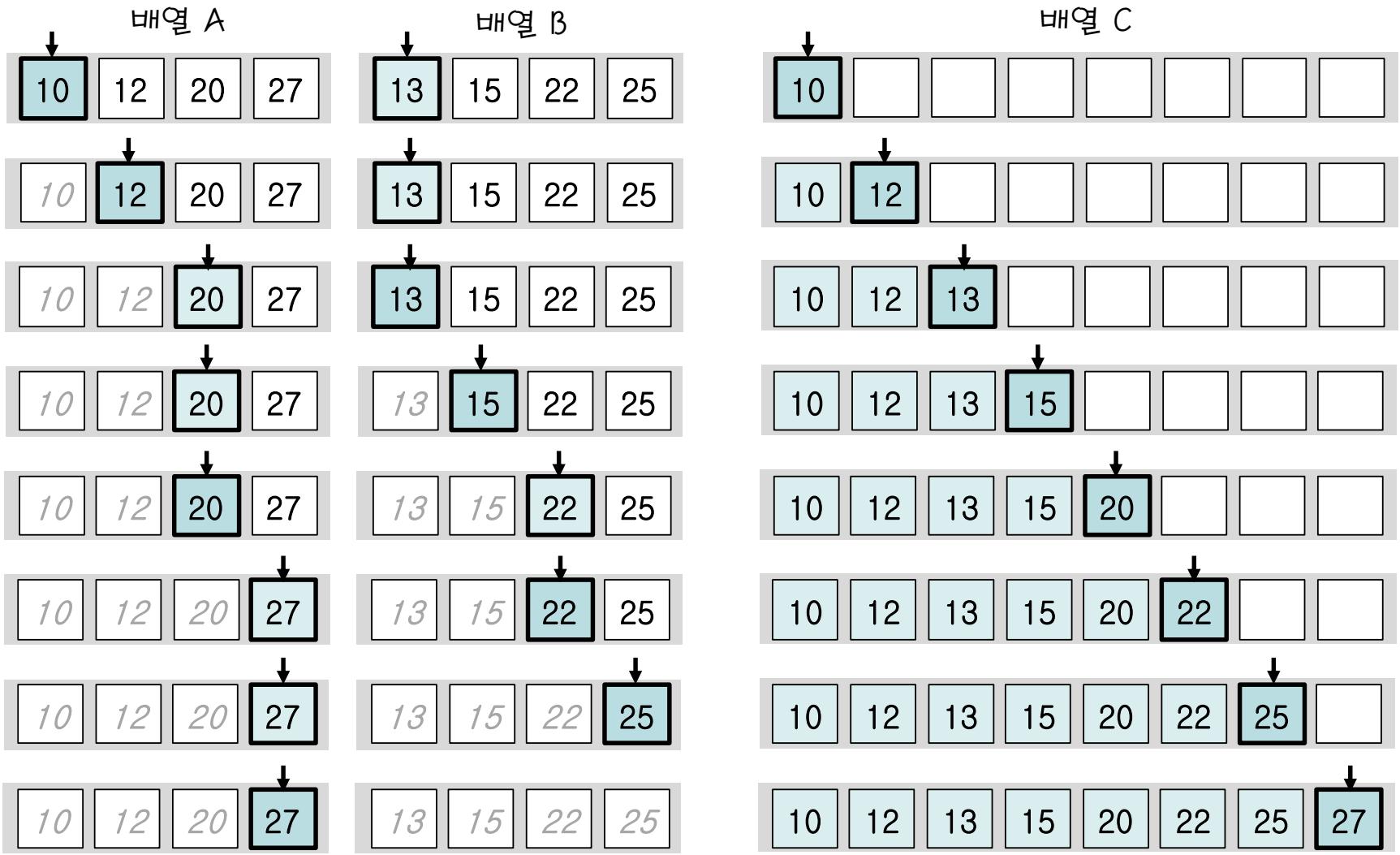
1.분할(Divide) :배열을 같은 크기의 2개의 부분 배열로 분할
2.정복(Conquer):부분배열을 정렬한다.부분배열의 크기가 충분히 작지 않으면 재귀호출을 이용하여 다시 분할 정복 기법 적용
3.결합(Combine):정렬된 부분배열을 하나의 배열에 통합

합병정렬 알고리즘

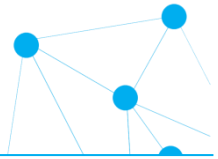

27 10 12 20 25 13 15 22
분할
27 10 12 20
25 13 15 22
분할
분할
27 10
12 20
25 13
15 22
분할
분할
분할
분할
27
10
12
20
25
13
15
22
병합
병합
병합
병합
10 27
12 20
13 25
15 22
병합
병합
10 12 20 27
13 15 22 25
병합
10 12 13 15 20 22 25 27

합병(=병합) 과정

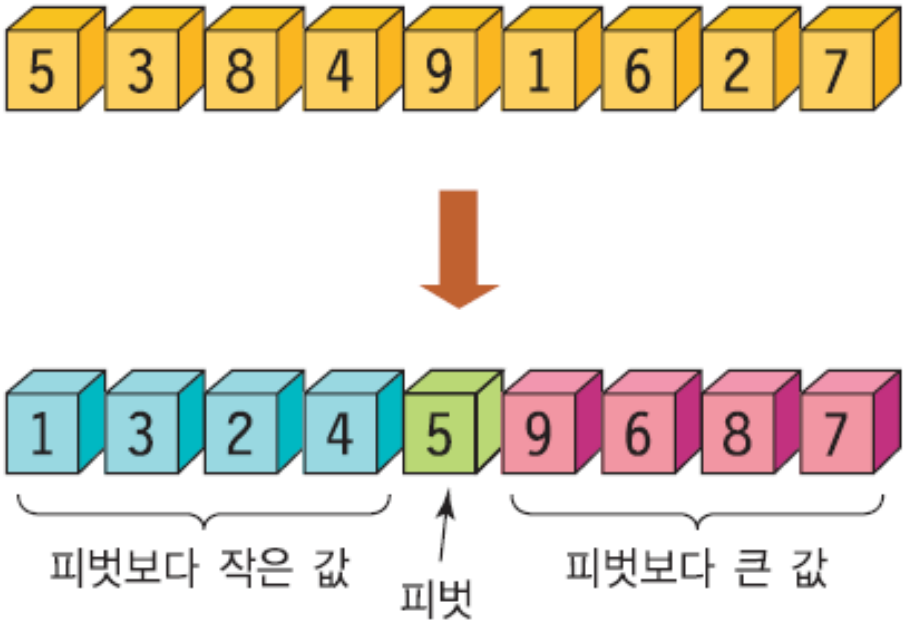



배열 A
배열 B
배열 C
10 12 20 27
13 15 22 25
10 12 20 27 13 15 22 25
10 12 20 27 13 15 22 25 10
10 12 20 27 13 15 22 25 10 12
10 12 20 27 13 15 22 25 10 12 13
10 12 20 27 13 15 22 25 10 12 13 15
10 12 20 27 13 15 22 25 10 12 13 15 20
10 12 20 27 13 15 22 25 10 12 13 15 20 22
10 12 20 27 13 15 22 25 10 12 13 15 20 22 25
10 12 20 27 13 15 22 25 10 12 13 15 20 22 25 27

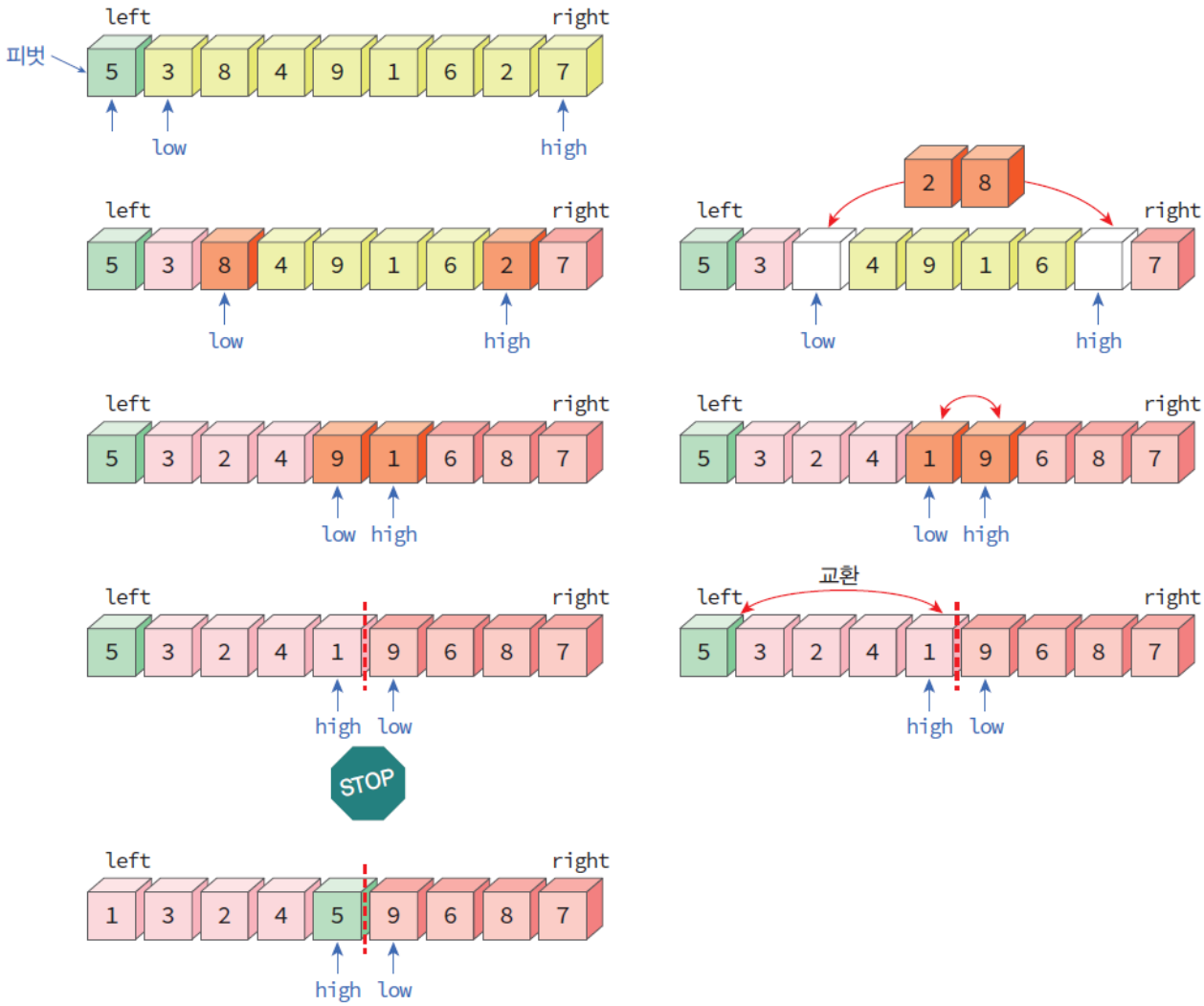
퀵정렬 (quick sort)



- 평균적으로 가장 빠른 정렬 방법
  - 분할 정복법 사용
  - 리스트를 2개의 부분리스트로 비균등 분할하고
  - 각각의 부분리스트를 다시 퀵 정렬함(순환 호출)



분할 과정



5를 피벗으로 선택
low ← left+1
high ← right

low를 피벗보다 큰 항목까지 이동
high를 피벗보다 작은 항목까지 이동

low와 high의 항목 교체

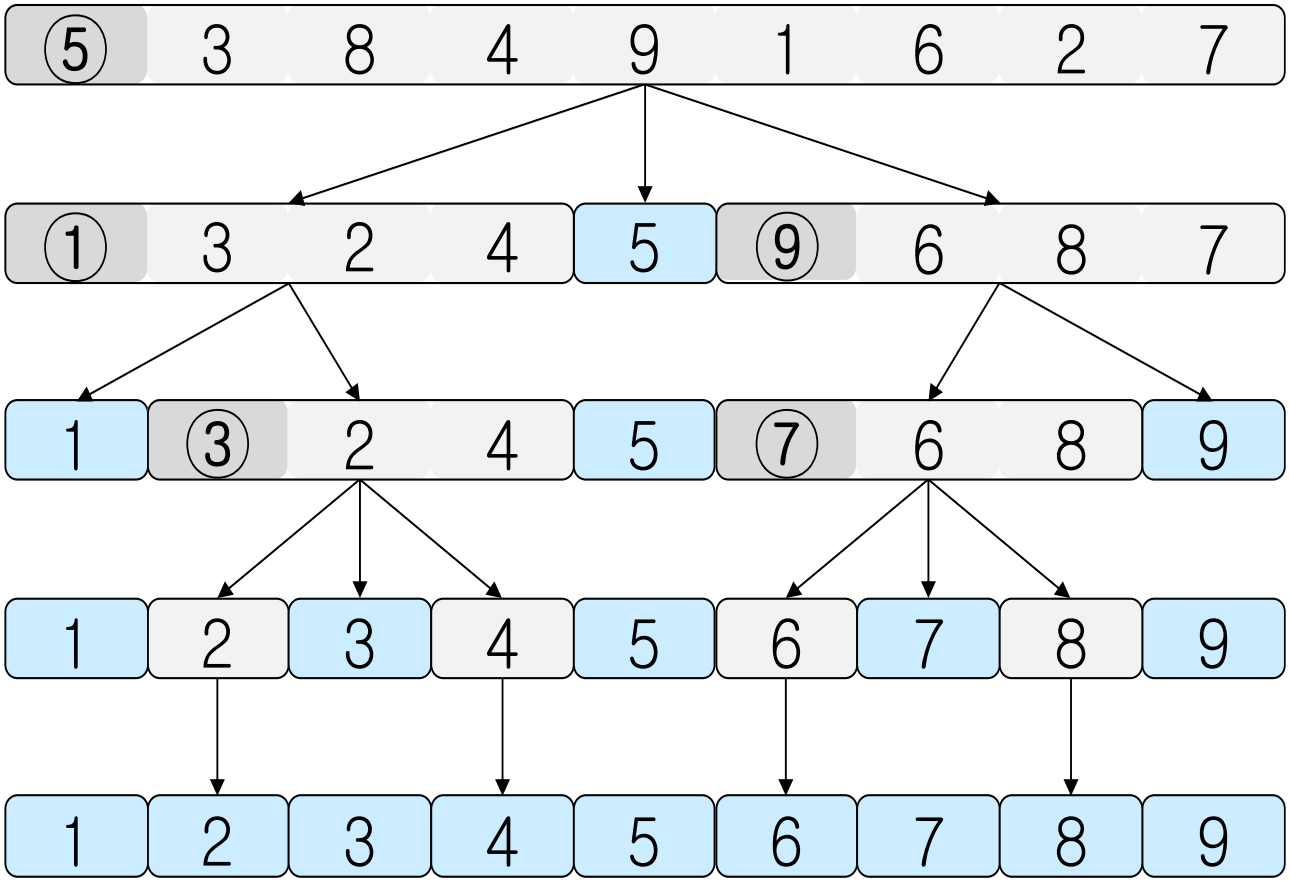
다시 진행
low를 피벗보다 큰 항목까지 이동
high를 피벗보다 작은 항목까지 이동

low와 high의 항목 교체

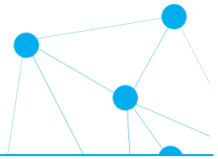
다시 진행
low와 high가 역전됨 → 종료

피벗과 high위치의 항목 교환

퀵정렬 전체과정



기수정렬(Radix Sort)

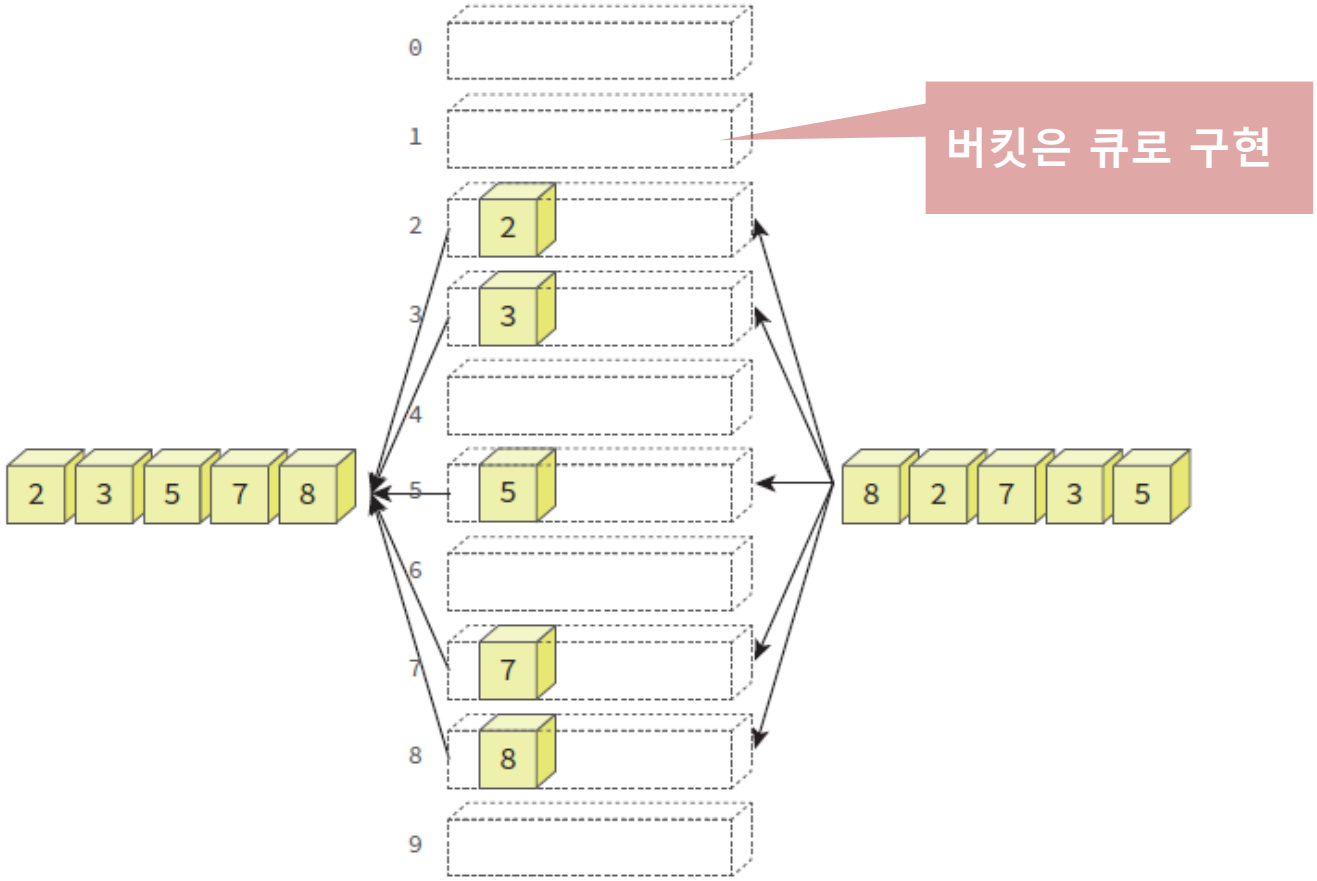


- 다른 정렬 방법들은 레코드를 비교하여 정렬 수행
- 기수 정렬은 레코드를 비교하지 않고 정렬 수행
  - 자리수의 값에 따라 정렬하는 방식

- 기수 정렬의 단점
  - 정렬할 수 있는 레코드의 타입 한정
    - 실수, 한글, 한자 등은 정렬 불가
  - 즉, 레코드의 키들이 동일한 길이를 가지는 숫자나 단순 문자(알파벳 등)이어야만 함

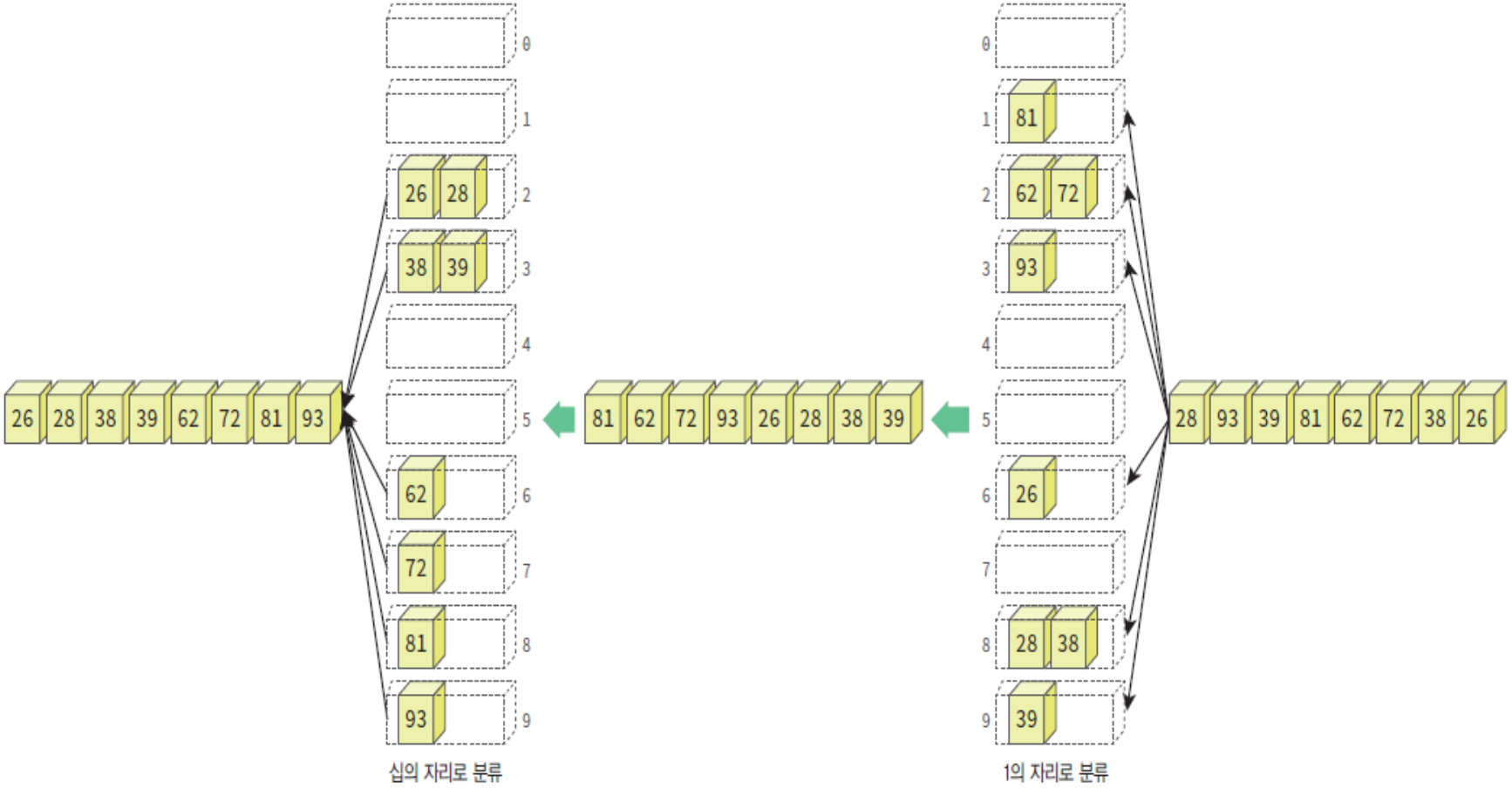
한 자릿수 기수정렬

- (8, 2, 7, 3, 5) 정렬의 예
  - 단순히 자리수에 따라 bucket에 넣었다가 꺼내면 정렬됨

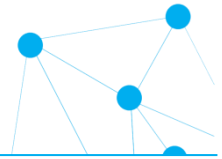


두 자릿수 기수정렬

- (28, 93, 39, 81, 62, 72, 38, 26)
  - 낮은 자릿수 분류 → 순서대로 읽음 → 높은 자릿수 분류



기수정렬



- 버킷의 개수는 키의 표현 방법과 밀접한 관계
  - 이진법을 사용한다면 버킷은 2개.
  - 알파벳 문자를 사용한다면 버킷은 26개
  - 십진법을 사용한다면 버킷은 10개

(예) 32비트의 정수의 경우, 8비트씩 나누면

→ 버킷은 256개로 늘어남.

→ 대신 필요한 패스의 수는 4로 줄어듦.