프로그래밍을 이용한 문제 해결
2일차 – 정렬과 자료구조

강영민
동명대학교 게임공학과

창의적 소프트웨어 융합 전문 인력 양성 사업단
2016년 1월 – 소프트웨어 역량 강화 프로그램

정렬

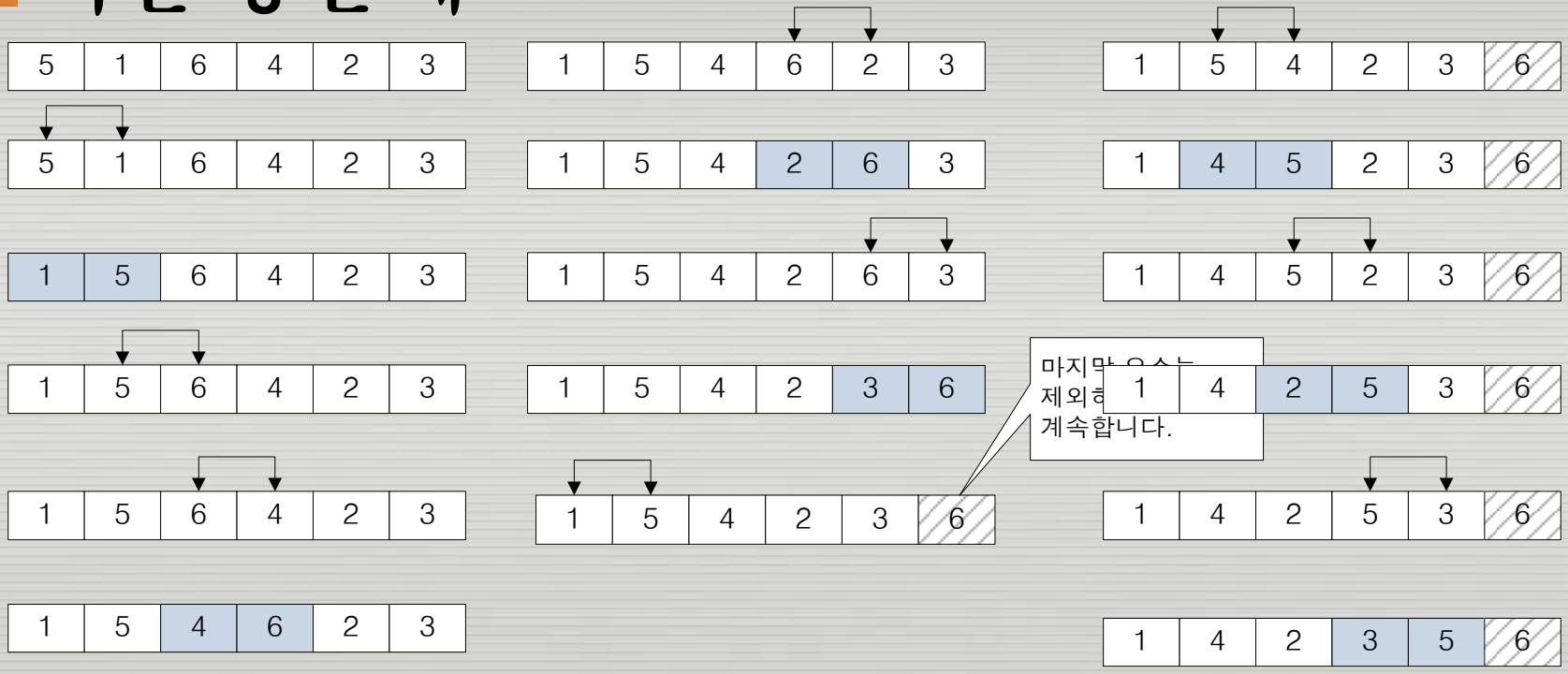
- 데이터를 순서에 따라 재배치
  - 방법
    - 매우 다양한 방법이 존재
  - 왜 하나의 방법이 아니라 여러 가지일까
  - 효율성이 다르다
  - 버블 정렬
    - 간단한 구현 / 큰 문제에 적용할 수 없음
  - 퀵소트 / 병합정렬 :
    - 구현은 다소 어려워도 매우 효율적이다
- 정렬은 가져다 쓰면 되지 않는가
  - 정렬을 다루면서 사용하는 기법은 다른 문제 풀이에 매우 중요한 개념을 제공

버블 정렬

- 정렬 과정이 물 속 깊은 곳에서 일어난 거품이 수면을 향해 올라오는 모습과 같다고 해서 붙여진 이름
- *데이터 집합을 순회하면서 집합 내의 이웃 요소들끼리의 교환을 통해 정렬*

버블 정렬

- 버블 정렬 예


5 1 6 4 2 3
5 1 6 4 2 3
1 5 6 4 2 3
1 5 6 4 2 3
1 5 6 4 2 3
1 5 4 6 2 3
1 5 4 6 2 3
1 5 4 2 6 3
1 5 4 2 6 3
1 5 4 2 3 6
1 5 4 2 3 6
1 5 4 6 2 3
1 5 4 2 3 6
1 5 4 2 3 6
1 4 5 2 3 6
1 4 5 2 3 6
1 4 2 5 3 6
1 4 2 5 3 6
1 4 2 3 5 6
마지막 요소는
제외하
계속합니다.

버블 정렬

- 버블 정렬의 성능

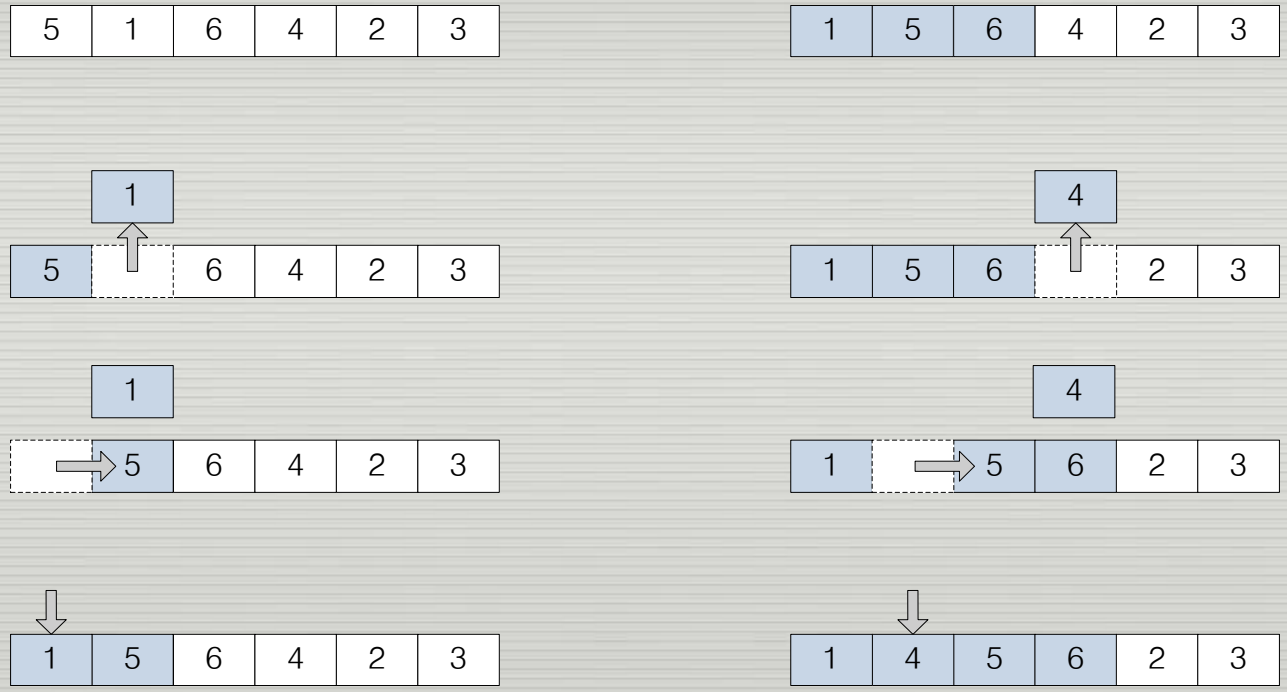
$$\text{버블 정렬의 비교 횟수} = (n-1)+(n-2)+(n-3)+\ldots+(n-(n-2))+(n-(n-1))$$

$$= (n-1)+(n-2)+(n-3)+\cdots+3+2+1$$

$$= (n-1)\times\frac{n}{2}=\frac{n(n-1)}{2}$$

삽입 정렬

- 삽입 정렬 예



삽입 정렬

- 삽입 정렬의 성능

$$\text{삽입 정렬의 비교 횟수} = 1 + 2 + \ldots + (n-2) + (n-1) = \frac{n(n-1)}{2}$$

- 삽입 정렬은 데이터 집합이 정렬되어 있는 경우에는 한번도 비교를 거치지 않는다.
- 데이터가 정렬되어 있는 최선의 경우와 역으로 정렬되어 있는 최악의 경우에 삽입 정렬이 수행하는 비교 횟수의 평균을 낸다면

$$\frac{(n(n-1)/2 + (n-1))}{2} = \frac{(n^2 + n - 2)}{4}$$

퀵 정렬

- “분할 정복(Divide and Conquer)”에 기반한 알고리즘

1. 데이터 집합 내에서 임의의 기준 요소를 선택하고, 기준 요소보다 작은 요소들은 순서에 관계없이 무조건 기준 요소의 왼편에, 큰 값은 오른편에 위치 시킵니다.

2. 기준 요소 왼편에는 기준 요소보다 작은 요소들이 모여 있고 오른편에는 큰 요소들이 모여 있겠죠? 이렇게 나눈 데이터 집합들을 다시 1에서와 같이 임의의 기준 요소를 선택하고 같은 방법으로 데이터 집합을 분할합니다.

3. 1과 2의 과정을 더 이상 데이터 집합을 나눌 수 없을 때까지 반복하면 정렬된 데이터 집합을 얻게 됩니다.

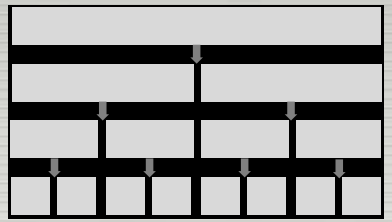
퀵 정렬

퀵 정렬 예


기준 요소로 선택
5 1 6 4 8 3 7 9 2
1 4 3 2 5 6 8 7 9
기준 요소로 선택
1 4 3 2 5 6 8 7 9
1 4 3 2 5 6 8 7 9
기준 요소로 선택
1 4 3 2 5 6 8 7 9
1 3 2 4 5 6 8 7 9
기준 요소로 선택
1 3 2 4 5 6 8 7 9
1 2 3 4 5 6 8 7 9
기준 요소로 선택
1 2 3 4 5 6 8 7 9
1 2 3 4 5 6 8 7 9
기준 요소로 선택
1 2 3 4 5 6 8 7 9
1 2 3 4 5 6 7 8 9

퀵 정렬

- 퀵 정렬의 성능 : 루프의 반복 횟수 대신 재귀 횟수를 이용하여 분석
  - 최선의 경우 :

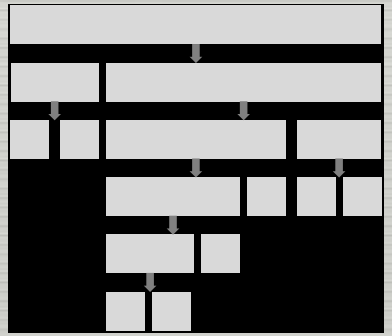


$$n \log_2 n$$

  - 최악의 경우 :

$$(n-1)+(n-2)+(n-3)+\cdots+3+2+1 = n \times \frac{n-1}{2} = \frac{n(n-1)}{2}$$

  - 평균의 경우 :

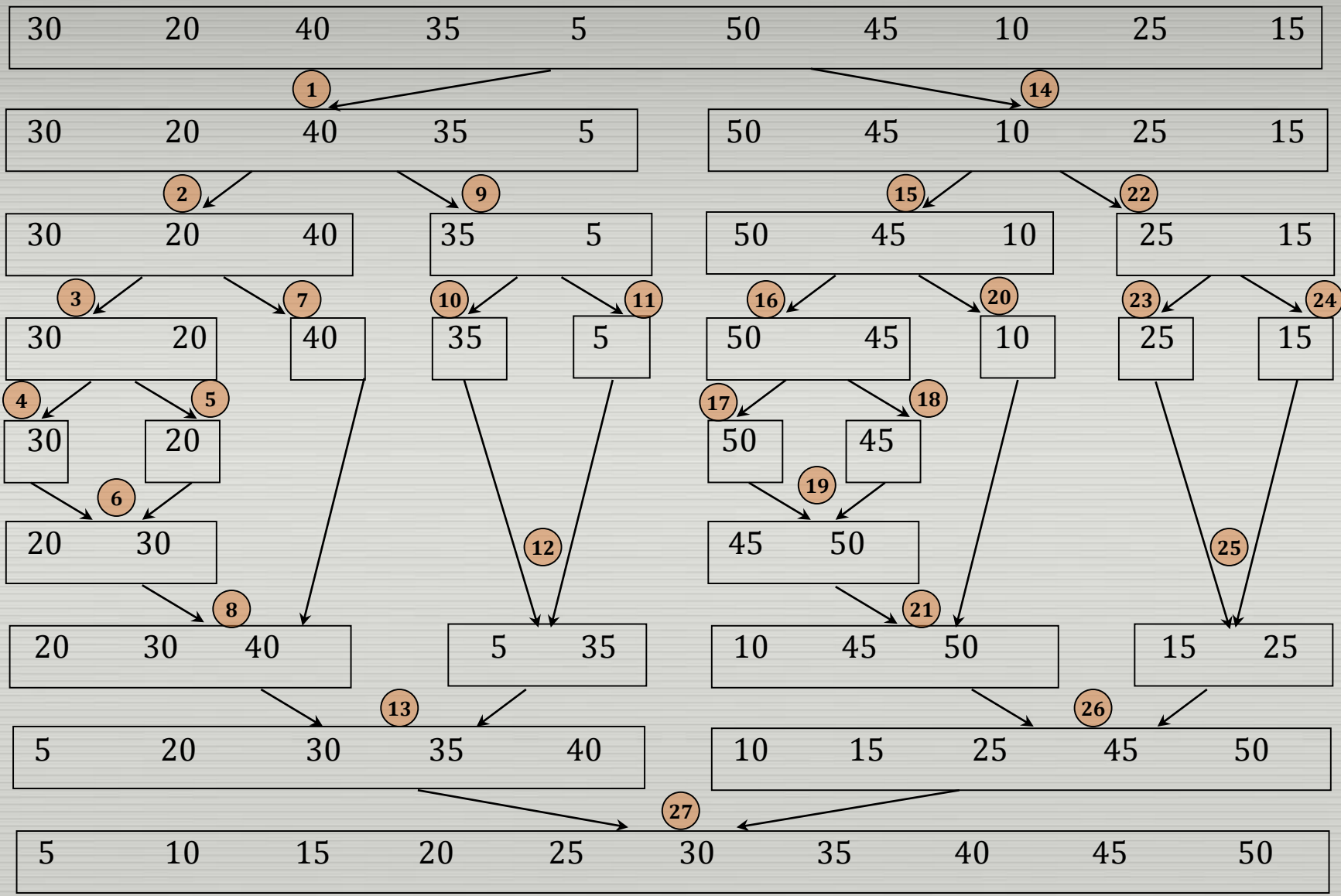


$$1.39\ n\log_2 n$$

합병정렬

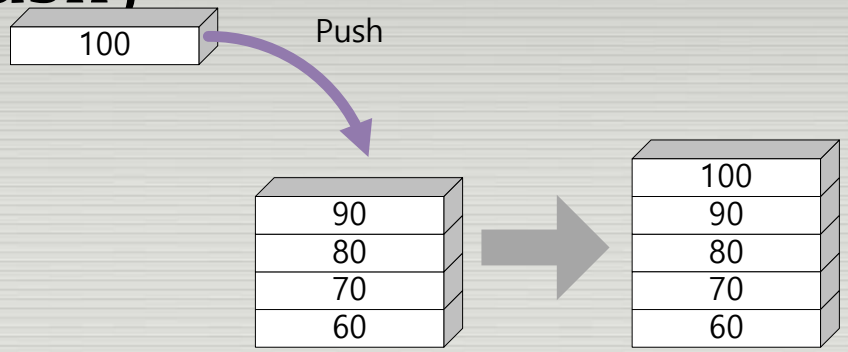
- 분할 정복 방식
- 동일한 크기의 두 부분배열로 분할하여 이 두 부분배열을 순환적으로 정렬한 후 합병하는 방식
- 최악의 수행시간이 O(n log n)이며 O(n)의 메모리가 별도로 필요

단계

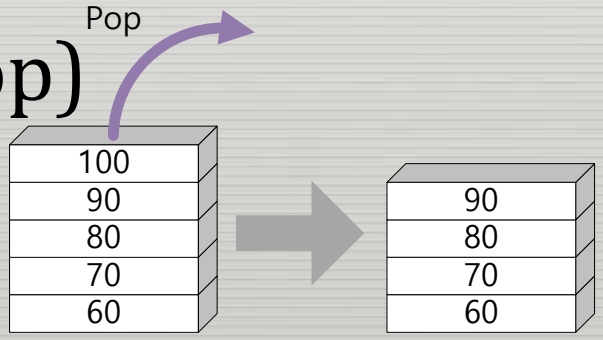

30 20 40 35 5 50 45 10 25 15
1
14
30 20 40 35 5
50 45 10 25 15
2
9
15
22
30 20 40
35 5
50 45 10
25 15
3
7
10
11
16
20
23
24
30 20
40
35
5
50 45
10
25
15
4
5
17
18
30
20
50
45
6
19
20 30
45 50
12
25
8
21
20 30 40
5 35
10 45 50
15 25
13
26
5 20 30 35 40
10 15 25 45 50
27
5 10 15 20 25 30 35 40 45 50

스택의 주요 기능: 삽입과 제거

- 삽입(Push)

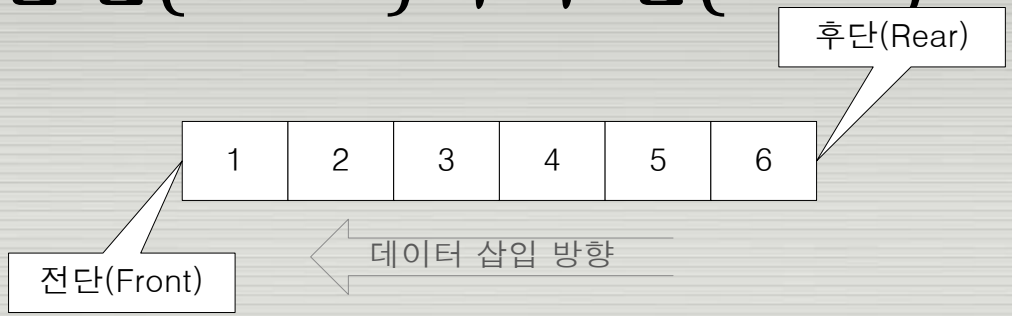


- 삭제(Pop)

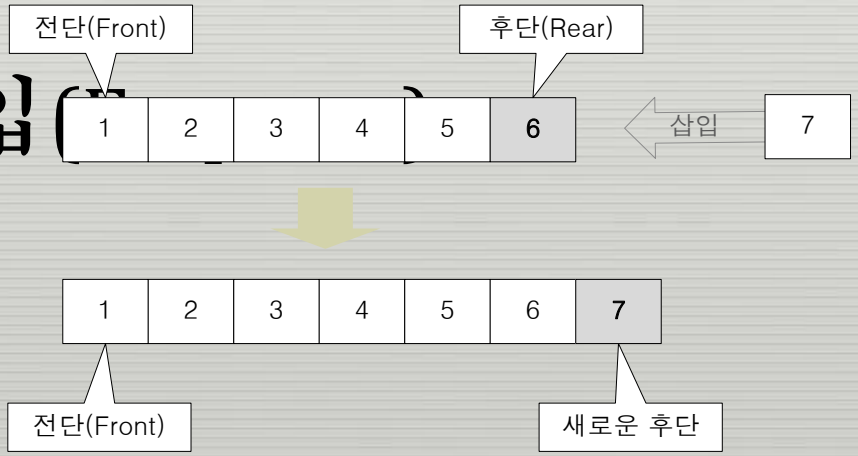


큐의 주요 기능: 삽입과 제거

- 전단(Front)과 후단(Rear)

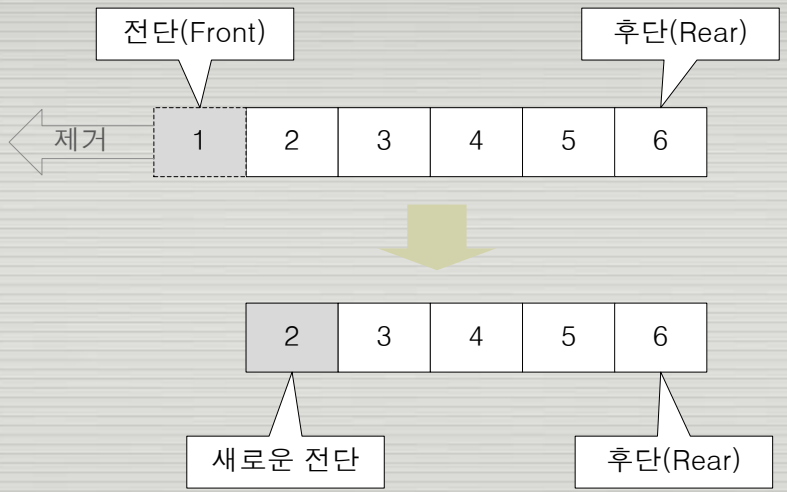


- 삽입(



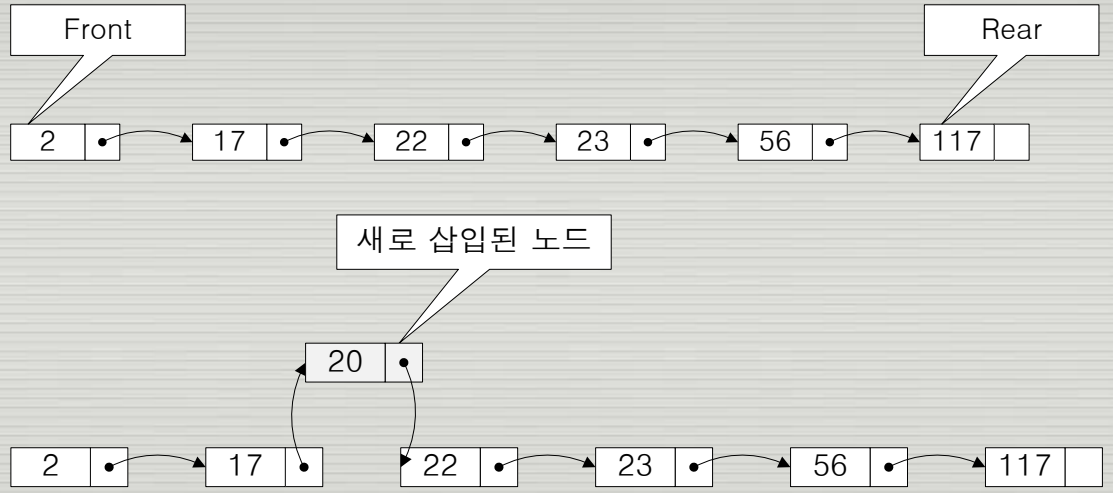
큐의 주요 기능: 삽입과 제거

- 제거(Dequeue)

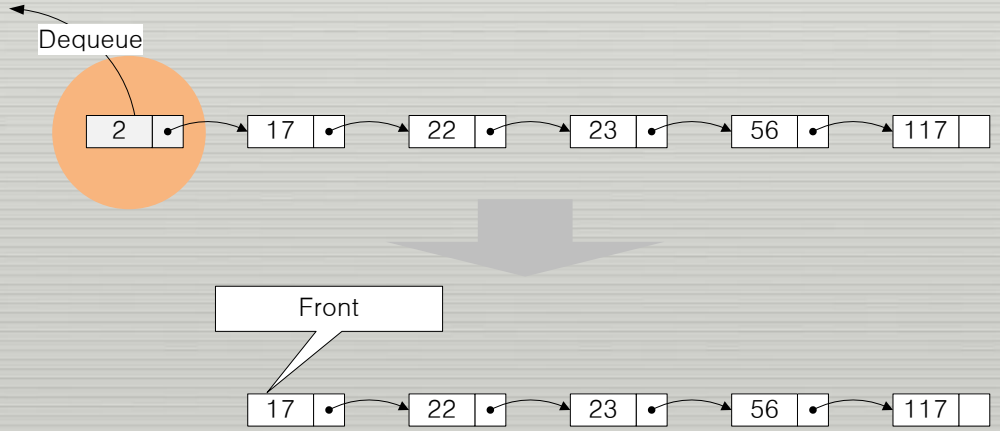


우선순위 큐

- 큐도 보통의 큐처럼 동작하지만 "우선순위" 속성을 갖는 데이터를 다룬다
- 삽입 연산

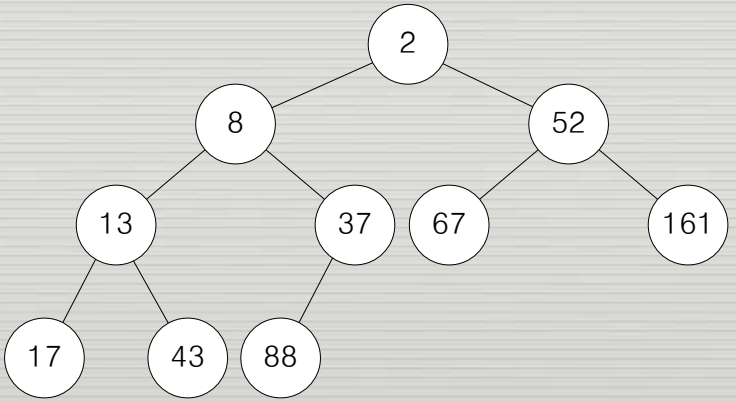


- 제거 연산



힙(heap)

- 힙 순서 속성(Heap Order Property)을 만족하는 완전 이진 트리
- 힙 순서 속성 : 트리 내의 모든 노드가 부모 노드보다 커야 한다는 규칙



- *"힙에서 가장 작은 데이터를 갖는 노드는 루트 노드이다."*

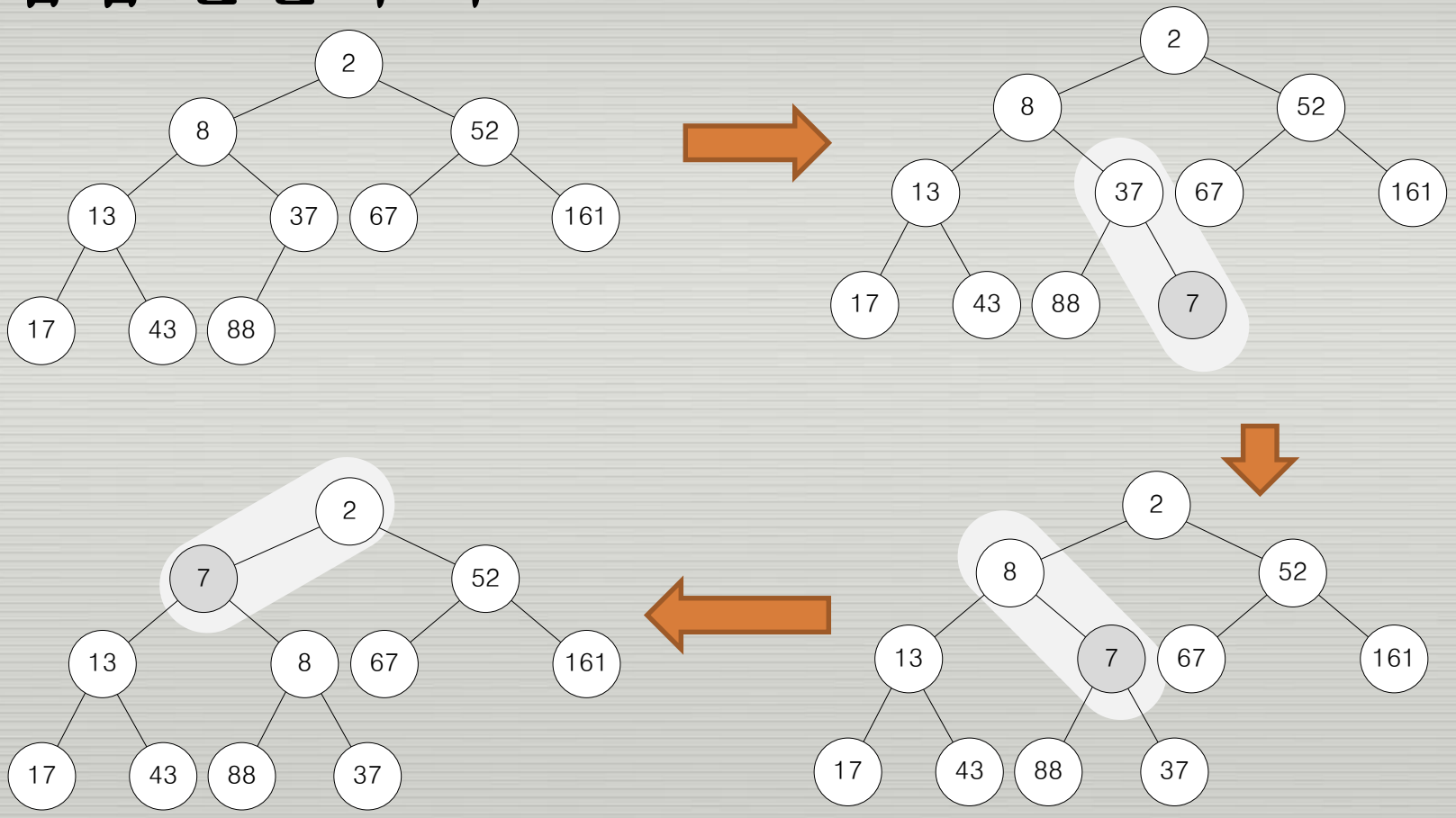
힙: 삽입(insert)

- 삽입 연산

1. 힙의 가장 최고 깊이, 최 우측에 새 노드를 추가한다. 물론 이 때 힙은 완전 이진 트리를 유지하도록 해야 한다.
2. 삽입한 노드를 부모 노드와 비교한다. 삽입한 노드가 부모 노드보다 크면 제 위치에 삽입된 것이므로 연산을 종료한다. 하지만 부모 노드보다 작으면 다음 단계를 진행한다.
3. 삽입한 노드가 부모 노드보다 작으면 부모 노드와 삽입한 노드의 위치를 서로 바꾼다. 바꾸고 나면 단계 2.를 다시 진행한다.

힙: 삽입 (insert)

- 삽입 연산의 예



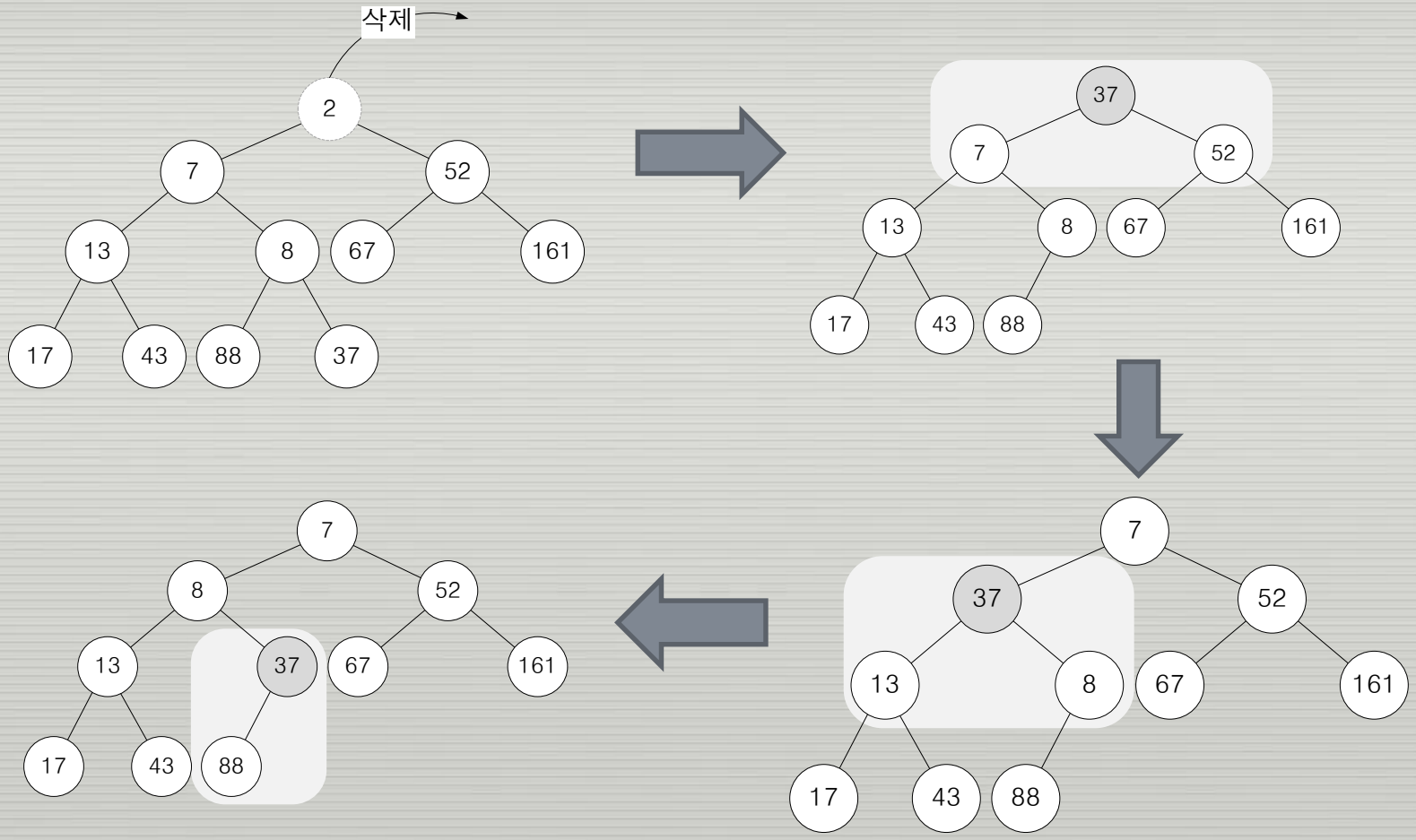
힙 – 삭제(delete)

- 최소값 삭제 연산
  - 루트 노드가 최소값 노드. 따라서 루트 노드를 삭제한 후 힙 순서 속성을 유지시키는 것이 삭제 연산의 관건.

1. 힙의 루트에 최고 깊이, 최 우측에 있던 노드를 루트 노드로 옮겨온다. 이 때 힙의 힙 순서 속성이 파괴된다. 이를 복원하기 위한 작업을 다음 단계에서부터 시작한다.
2. 옮겨온 노드의 양쪽 자식을 비교하여 작은 쪽 자식과 위치 교환을 한다. 힙 순서 속성이 지켜지려면 부모 노드는 양족 자식보다 작은 값을 가져야 하기 때문이다.
3. 옮겨온 노드가 더 이상 자식이 없는 잎노드가 되거나 양쪽 자식보다 작은 값을 갖는 경우에는 삭제 연산을 종료한다. 그렇지 않은 경우에는 단계 2를 반복한다.

힙 – 삭제(delete)

- 최소값 삭제 연산의 예



힙 – 배열을 이용한 구현

- 배열을 이용한 힙의 구현 (힙은 완전 이진 트리)
  - 깊이 0의 노드는 배열의 0번 요소에 저장.
  - 깊이 1의 노드(모두 2개)는 배열의 1~2번 요소에 저장.
  - 깊이 2의 노드(모두 4개)는 배열의 3~6번 요소에 저장.
  - **"깊이 n의 노드(2^n개)는 배열의 $2^n+1 \sim 2^{n+1}-2$ 번 요소에 저장."**