06 강

컴퓨터과학 개론

# 알고리즘 (2)

컴퓨터과학과 이관용 교수

# 학습목차

1. 정렬 알고리즘: 퀵정렬, 합병정렬
2. 순차탐색, 이진 탐색
3. 이진 탐색 트리

01

# 정렬 알고리즘:

## 퀵 정렬, 합병 정렬

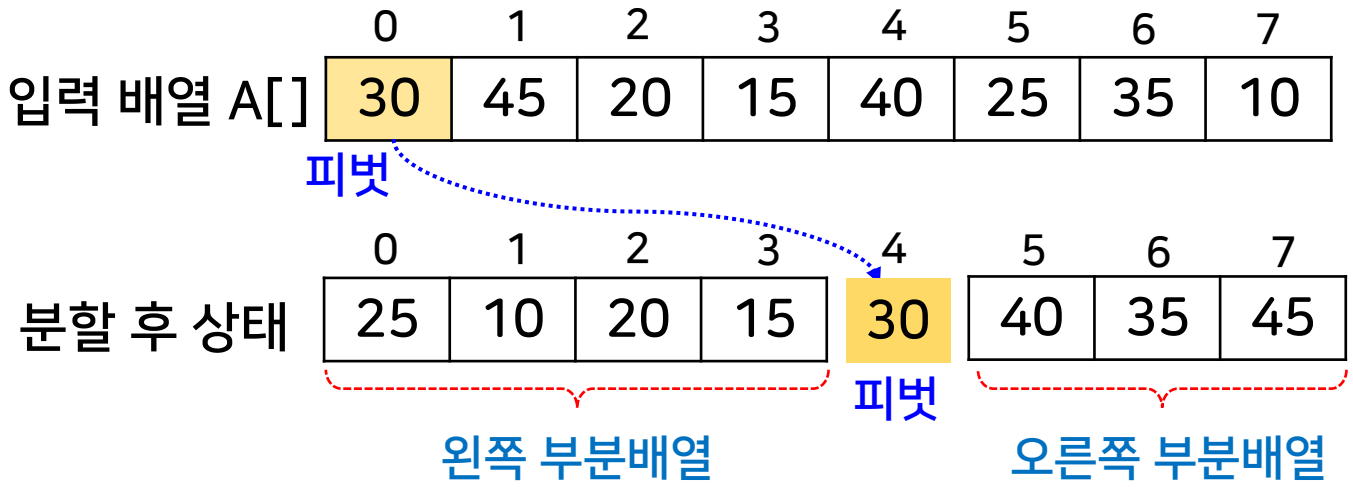
# 퀵 정렬

- **특정 데이터를 기준으로 입력 배열을 두 부분배열로 분할하고, 각 부분배열에 대해서 독립적으로 퀵 정렬을 순환적으로 적용**
  - 평균적으로 가장 좋은 성능의 비교 기반 알고리즘 → O(nlogn)

- **피벗 pivot, 분할원소**
  - 두 개의 부분배열로 분할할 때 기준이 되는 데이터
  - 보통 주어진 배열의 첫 번째 원소로 지정

# 퀵 정렬

## 피벗이 제자리를 잡도록 하여 정렬하는 방식

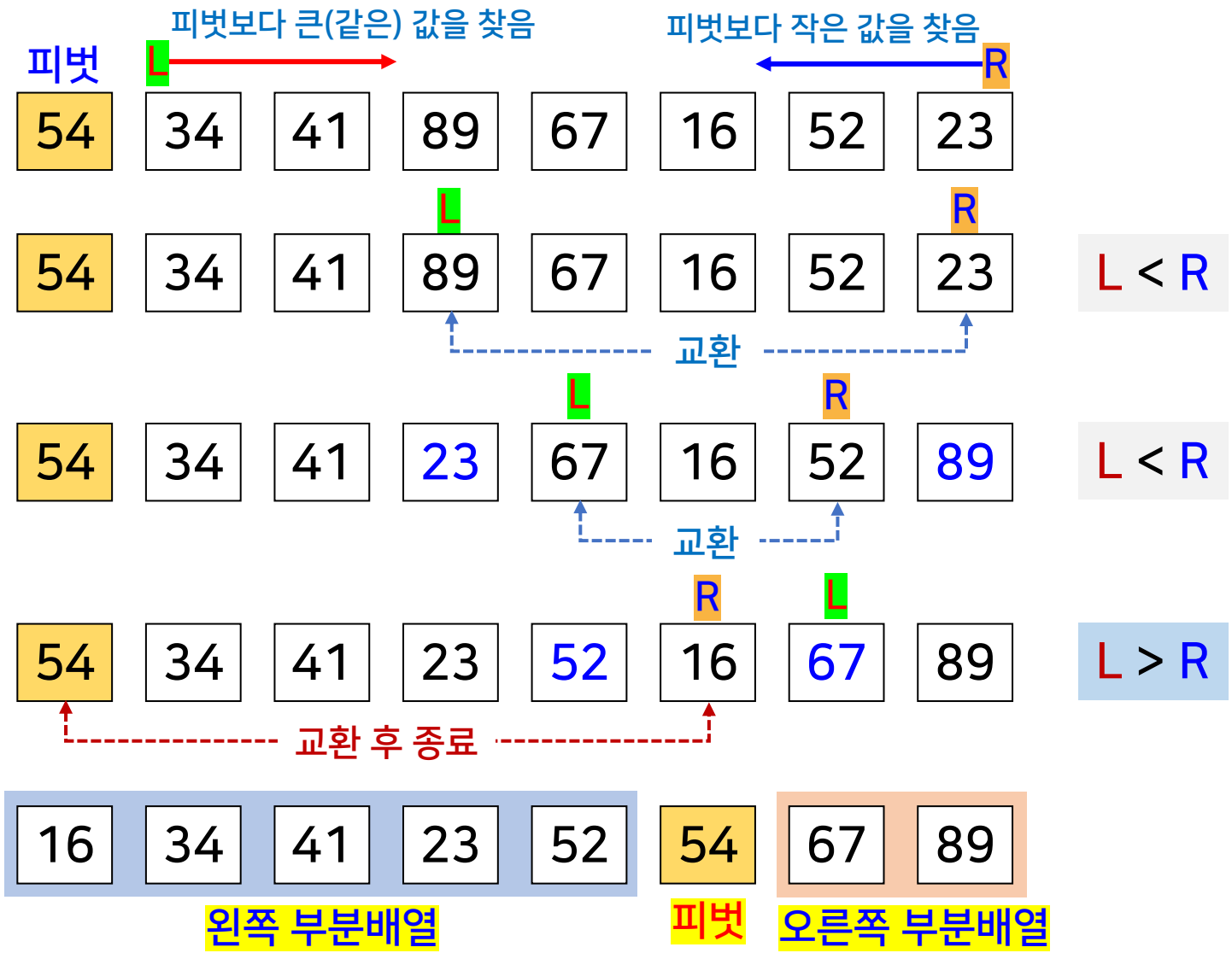



왼쪽 부분배열의 모든 데이터 < 오른쪽 부분배열에서 가장 작은 데이터

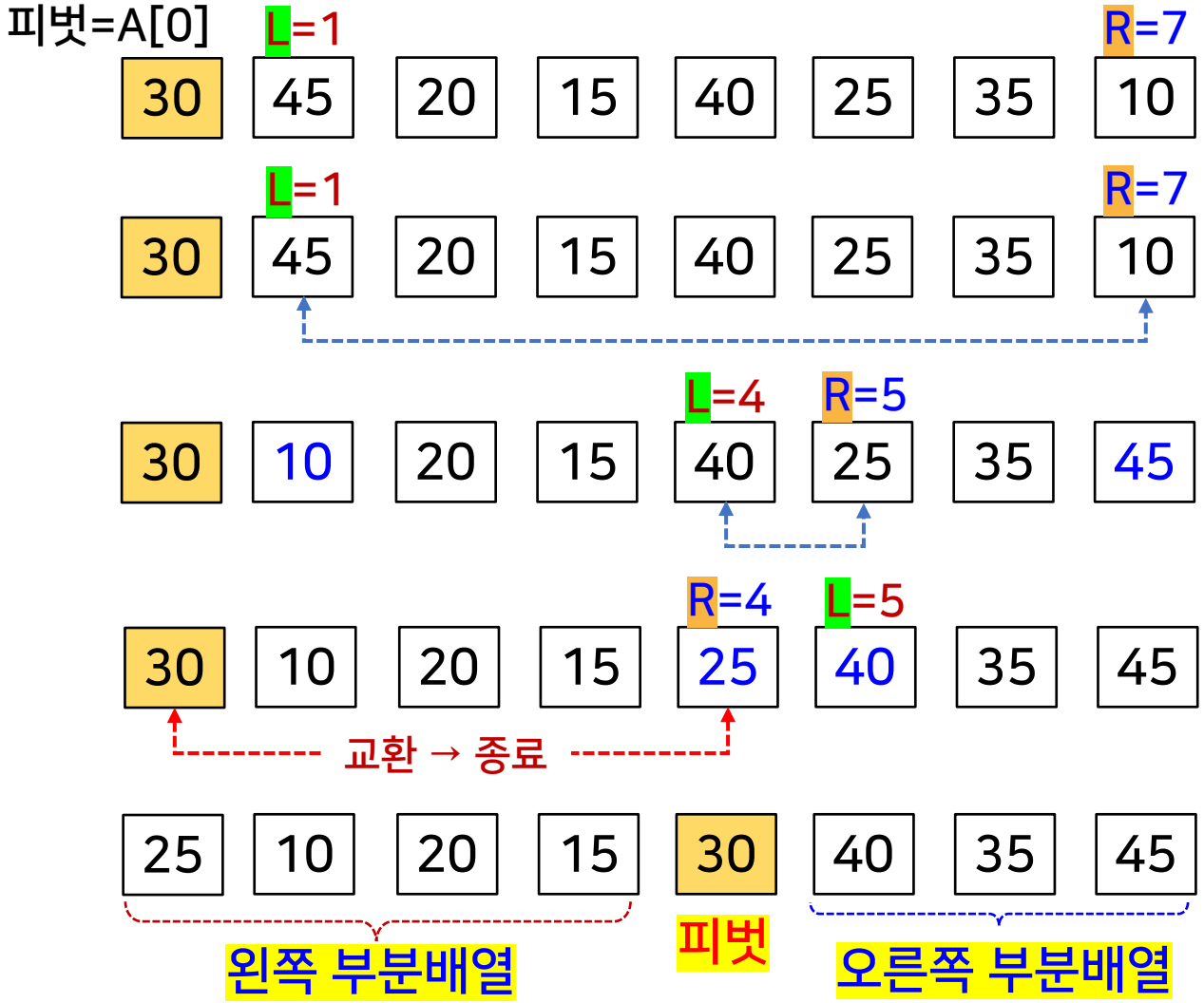
왼쪽 부분배열에서 가장 큰 데이터 < 오른쪽 부분배열의 모든 데이터

왼쪽 부분배열의 모든 값 < 피벗 < 오른쪽 부분배열의 모든 값

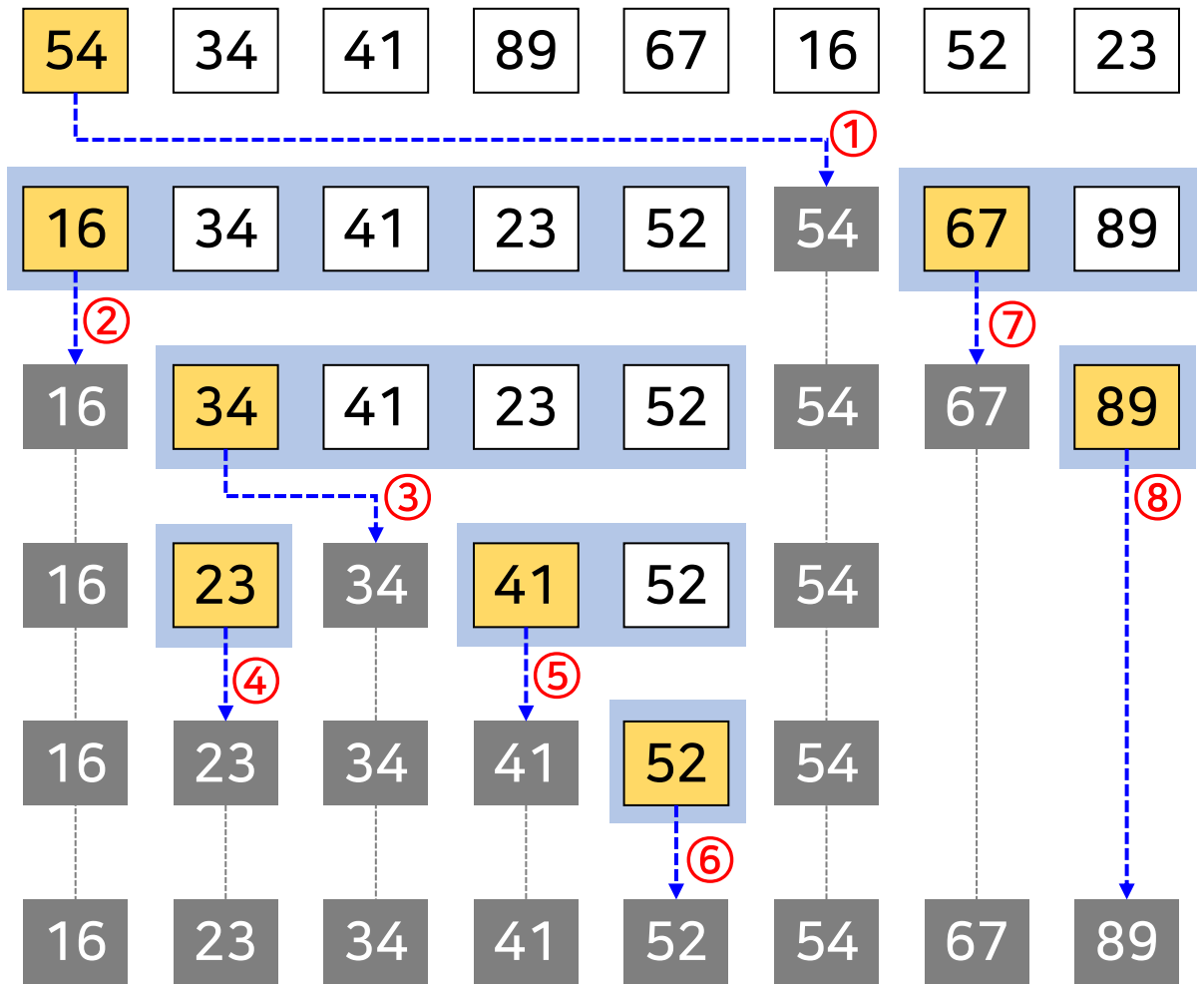
# 분할 과정_1


피벗보다 큰(같은) 값을 찾음
피벗보다 작은 값을 찾음
피벗
L
R
54 34 41 89 67 16 52 23
L
R
54 34 41 89 67 16 52 23
L < R
교환
L
R
54 34 41 23 67 16 52 89
L < R
교환
R
L
54 34 41 23 52 16 67 89
L > R
교환 후 종료
16 34 41 23 52
54
67 89
왼쪽 부분배열
피벗
오른쪽 부분배열

# 분할 과정_2


피벗=A[0]
L=1
R=7
30 45 20 15 40 25 35 10
L=1
R=7
30 45 20 15 40 25 35 10
L=4
R=5
30 10 20 15 40 25 35 45
R=4
L=5
30 10 20 15 25 40 35 45
교환 → 종료
25 10 20 15 30 40 35 45
왼쪽 부분배열
피벗
오른쪽 부분배열

# 퀵 정렬의 전체적인 수행 과정



# 퀵 정렬의 특징

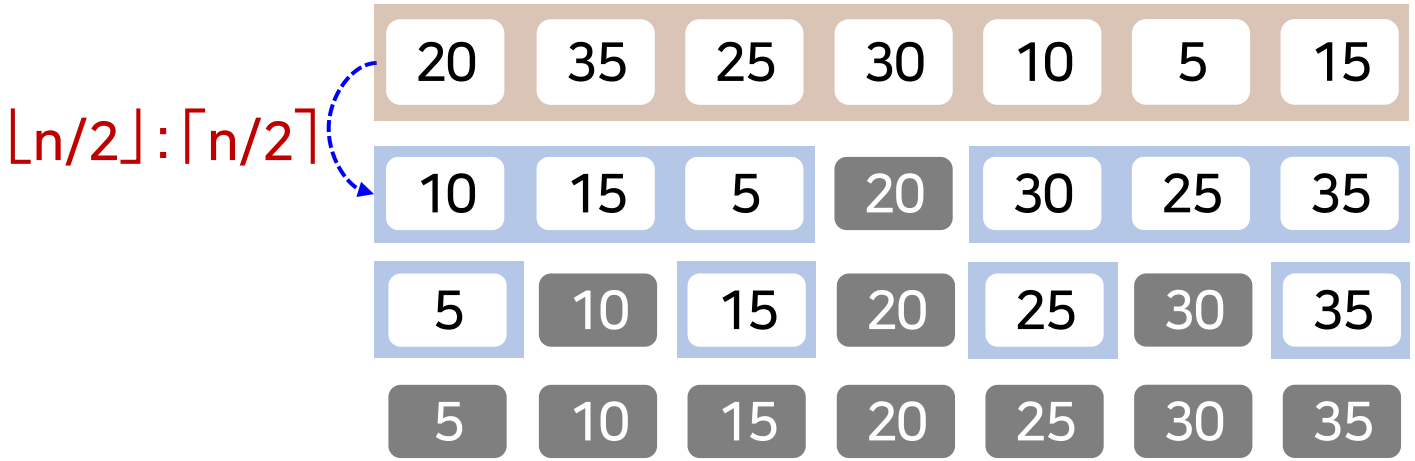
## 분할정복 방법을 적용한 알고리즘

- 분할
  - 정렬할 n개의 데이터를 피벗을 중심으로 두 개의 부분배열로 분할
- 정복
  - 두 부분배열 각각에 대해 퀵 정렬을 순환적으로 적용하여 두 부분배열을 정렬
- 결합
  - 필요 없음

# 퀵 정렬의 특징

## 성능

- 분할 과정의 수행 시간 → O(n)
- 평균 수행 시간 → O(nlogn)
- 최선 수행 시간 → O(nlogn)
  - 피벗을 중심으로 항상 동일한 크기의 두 개의 부분배열로 분할되는 경우



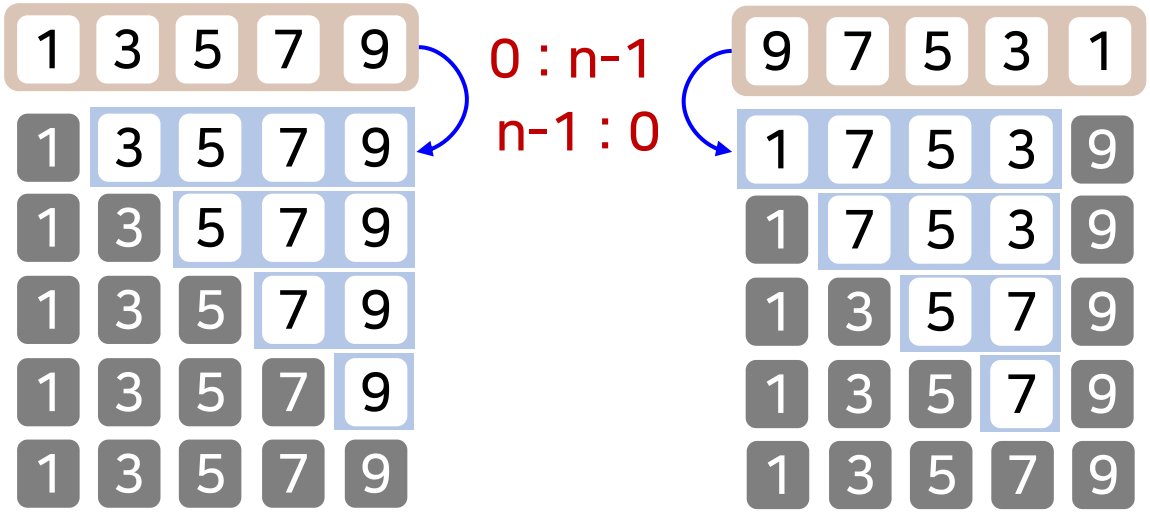
# 퀵 정렬의 특징

## 성능

- 최악 수행 시간 → $O(n^2)$

피벗 선택의 임의성만 보장되면
평균 수행 시간을 보일 가능성이 높음

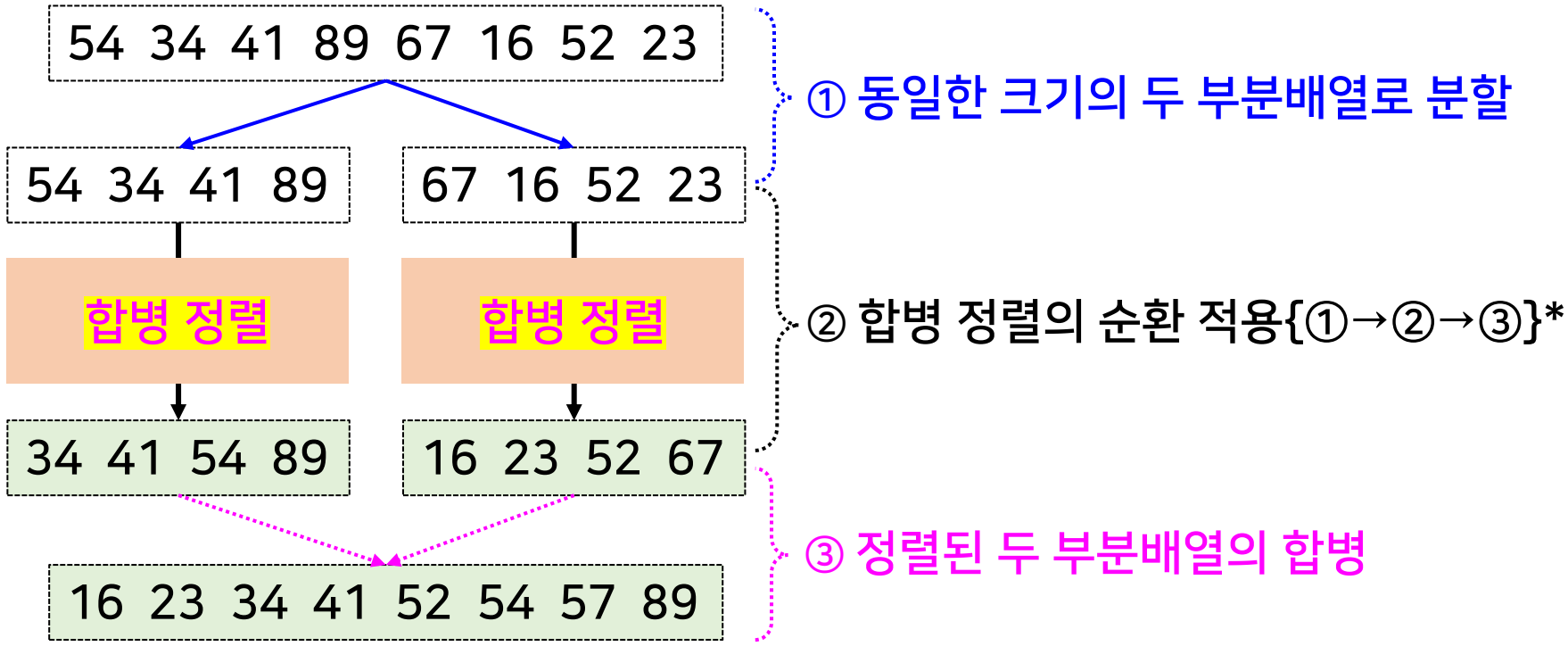
  - 피벗만 제자리를 잡고 나머지 모든 데이터가 하나의 부분배열로 분할되는 경우



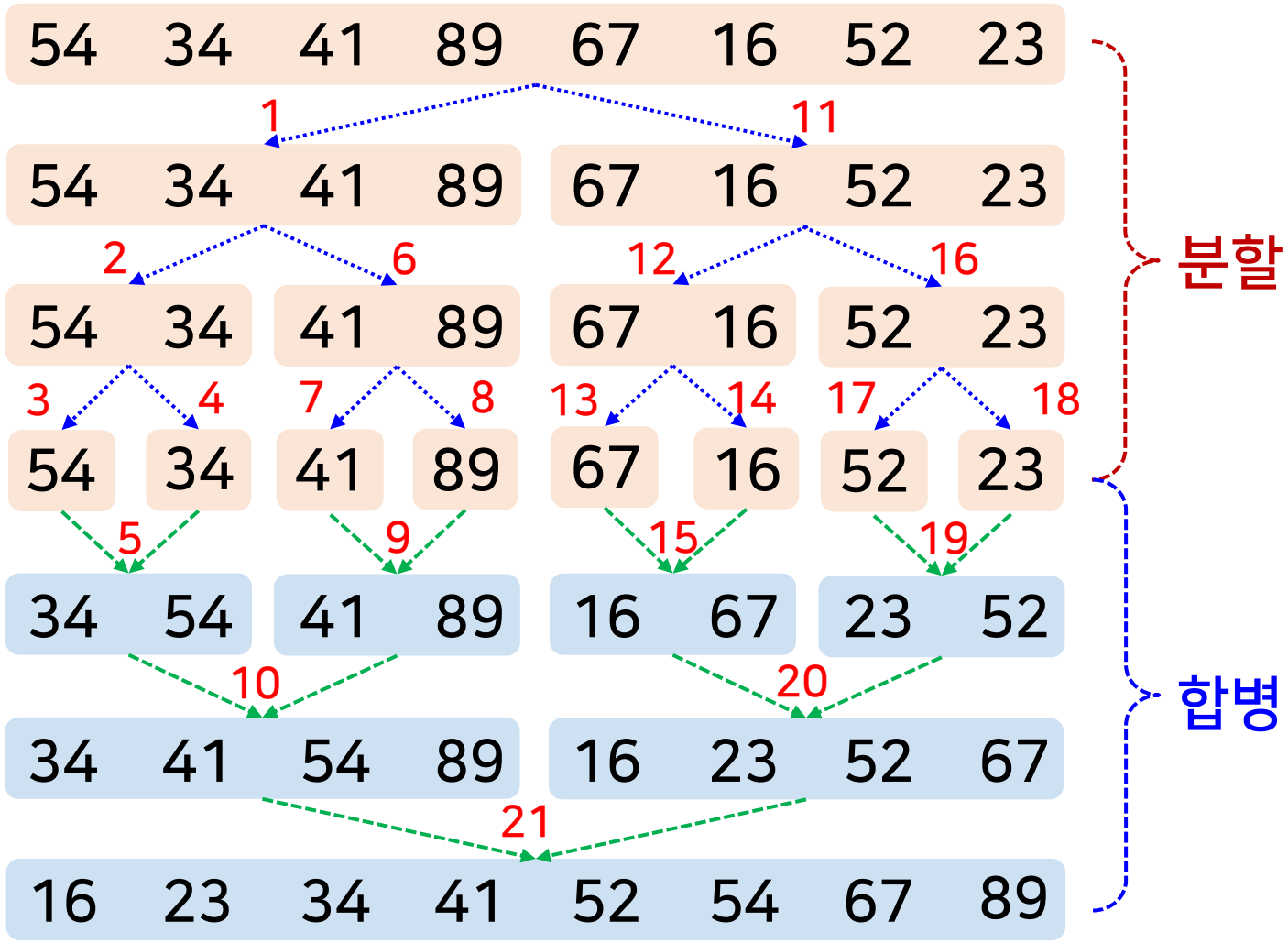

  - 피벗이 배열에서 항상 최댓값이나 최솟값이 되는 경우
  - 입력 데이터가 이미 정렬된 경우 AND 피벗을 첫 번째 원소로 지정한 경우

# 합병 정렬

- **동일한 크기의 두 개의 부분배열로 분할하고,**
  **각 부분배열을 순환적으로 정렬한 후,**
  **정렬된 두 부분배열을 합병해서 하나의 정렬된 배열을 만드는 방식**

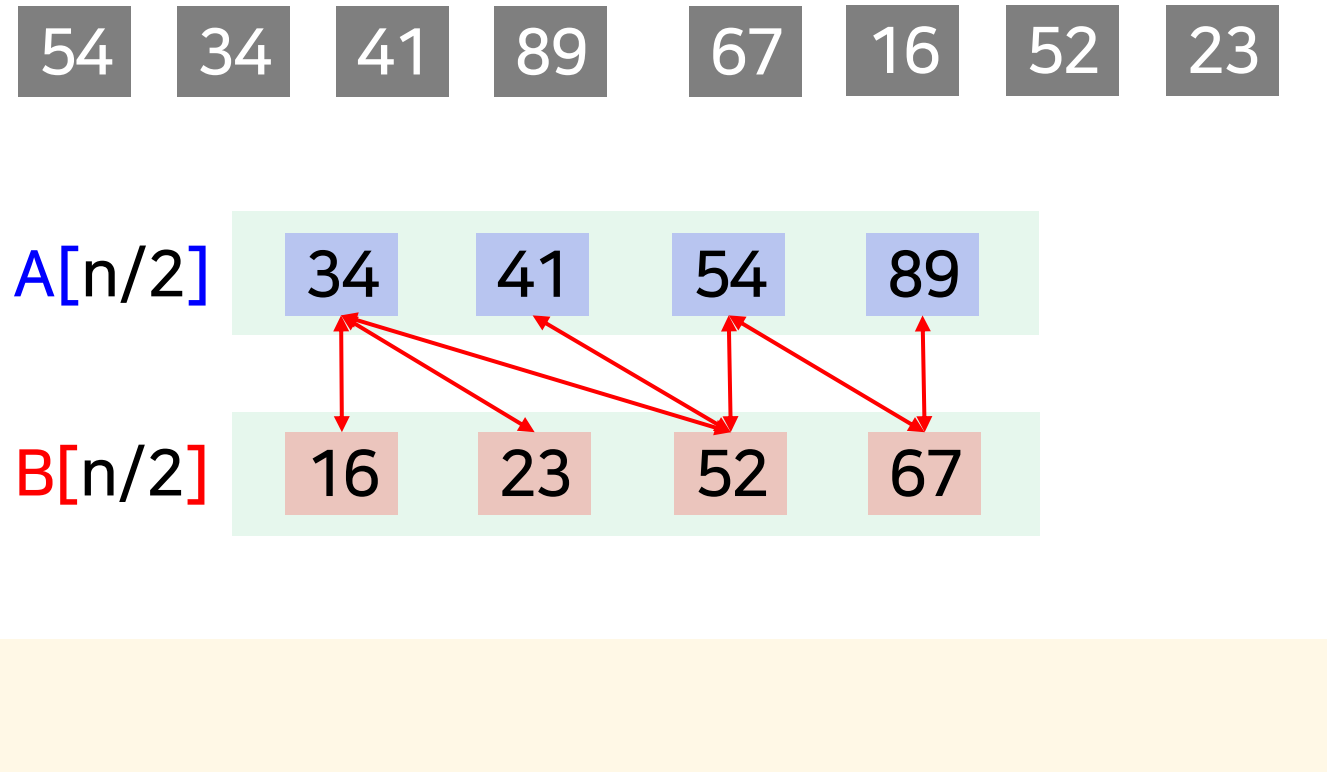


# 합병 정렬의 전체적인 수행 과정



# 합병 과정

- 정렬된 두 부분배열을 하나의 정렬된 배열로 만드는 과정



# 합병 정렬의 특징

## 분할정복 방법을 적용한 알고리즘

- 분할
  - 정렬할 n개의 데이터를 n/2개의 데이터를 갖는 두 부분배열로 분할
- 정복
  - 두 부분배열에 대해 합병 정렬을 각각 순환적으로 적용하여 정렬
- 합병
  - 정렬된 두 부분배열을 합병하여 하나의 정렬된 배열을 만듦

## 최선, 최악, 평균 수행 시간 → O(nlogn)

02

# 순차 탐색, 이진 탐색

# 탐색

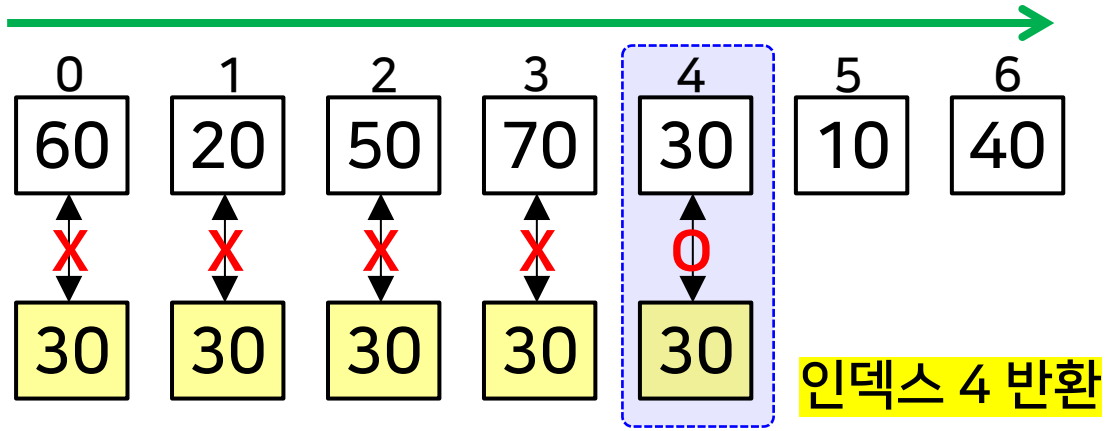
## 주어진 데이터 집합에서 원하는 값을 가진 데이터를 찾는 작업

- 순차 탐색 sequential search → O(n)
- 이진 탐색 binary search → O(logn)
- 이진 탐색 트리 binary search tree → 평균 O(logn), 최악 O(n)

# 순차 탐색

- **리스트 형태로 주어진 데이터를 처음부터 하나씩 차례대로 비교하여 원하는 데이터를 찾는 방법**

배열 A[ ]에서 탐색키 30의 탐색 과정



# 순차 탐색의 특징

## 탐색 성능 → O(n)

- 실패하는 경우의 비교 횟수 → $n$번
- 성공하는 경우의 비교 횟수 → $1 \sim n$번 → 평균 $\frac{n+1}{2}$번

## 모든 리스트(배열, 연결 리스트)에 적용 가능

- 데이터가 키값과 관련해서
  아무런 순서 없이 단순하게 연속적으로 저장
  → 데이터가 정렬되지 않은 경우에 적합

# 이진 탐색

- **정렬된 입력 배열에 대해서**
  **주어진 데이터를 절반씩 줄여가면서 원하는 데이터를 찾는 방법**
  - 분할정복 방법을 적용한 알고리즘

- **탐색 방법**

$$Mid = \left\lfloor \frac{Left + Right}{2} \right\rfloor$$

  - 입력 배열의 가운데 값 A[ Mid ] 과 탐색키 $key$를 비교
    - $key$ = A[ Mid ] → 탐색 성공 (배열의 인덱스 Mid 반환)
    - $key$ < A[ Mid ] → 이진 탐색(원래 크기의 ½인 왼쪽 부분배열)
    - $key$ > A[ Mid ] → 이진 탐색(원래 크기의 ½인 오른쪽 부분배열)

탐색을 반복할 때마다 대상 원소의 개수가 ½씩 감소

# 이진 탐색의 탐색 과정

Left=0 Right=8

| 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|

탐색키 key=35

Mid=4

A[Mid] < key이므로 오른쪽 부분배열을 다시 탐색

Left=5 Right=8

| 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|

Mid=6

key < A[Mid]이므로 왼쪽 부분배열을 다시 탐색

Left=Right=5

| 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|

Mid=5

key = A[Mid]이므로 탐색 성공(Mid=5 반환)

# 이진 탐색의 특징

- **성능** → O(logn)
  - 한 번 탐색할 때마다 탐색 대상이 되는 데이터의 개수가 ½씩 감소
- **데이터가 이미 정렬된 경우에만 적용 가능**
- **삽입/삭제 연산 시 정렬 상태의 유지를 위해 데이터 이동이 발생**
  - 삽입/삭제와 같은 동적 연산이 많은 응용에는 부적합

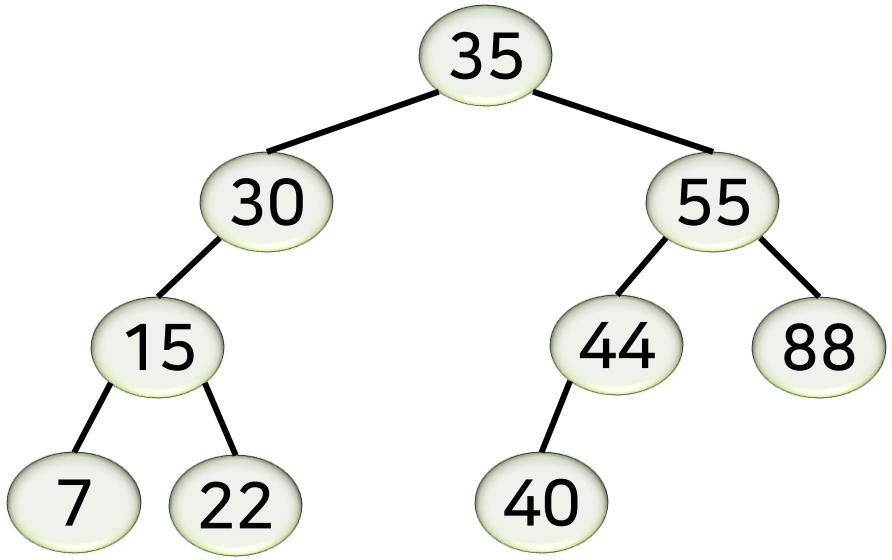
03

# 이진 탐색 트리

# 이진 탐색 트리

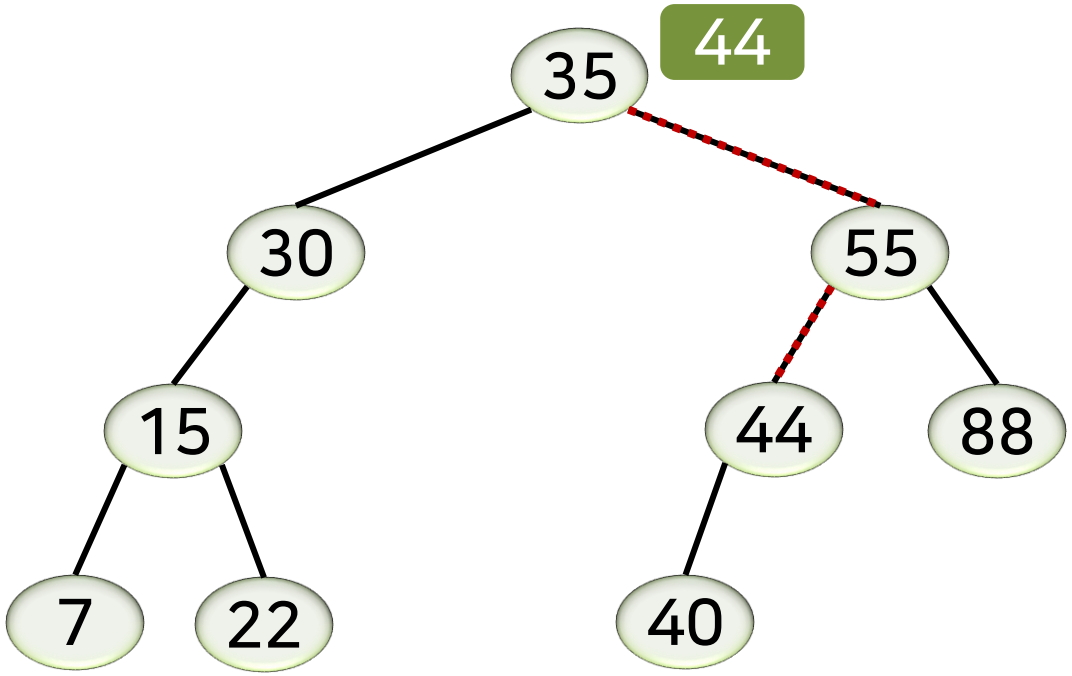
## 다음의 두 성질을 만족하는 이진 트리

- 각 노드의 왼쪽 서브트리에 있는 모든 키값은 그 노드의 키값보다 작다.
- 각 노드의 오른쪽 서브트리에 있는 모든 키값은 그 노드의 키값보다 크다.



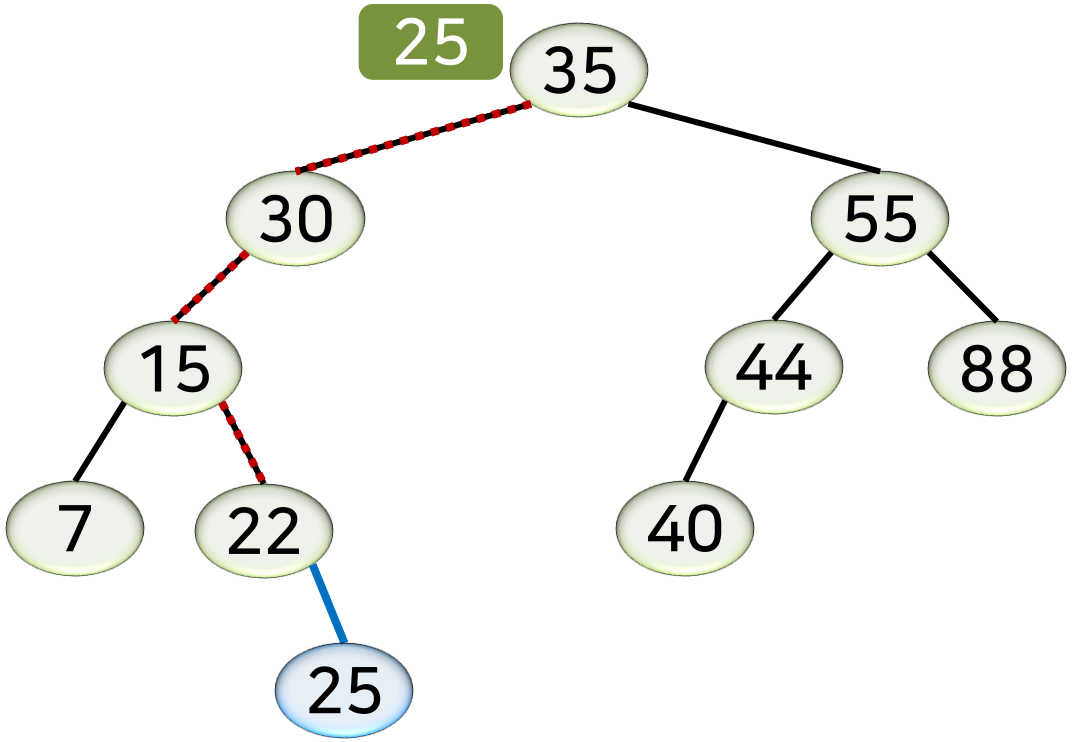
# 탐색 연산

- 루트 노드에서부터 키값의 비교를 통해
  왼쪽 또는 오른쪽 서브트리를 따라 이동하면서 데이터를 찾음



# 삽입 연산

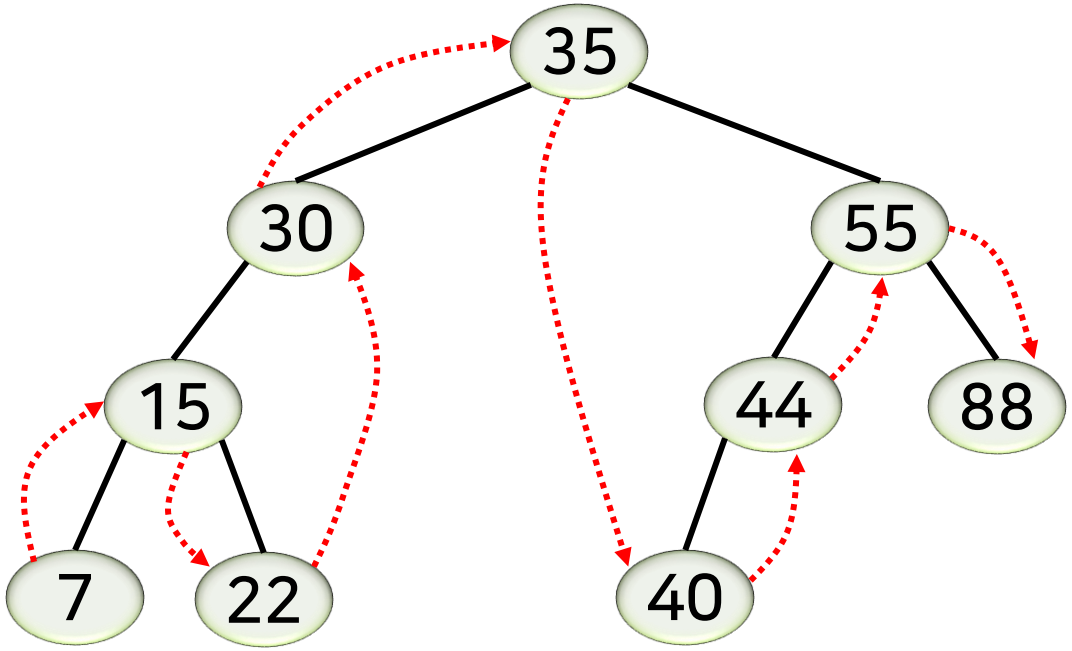
- **삽입할 데이터를 탐색한 후,**
  **탐색이 실패한 위치에 새로운 노드를 자식 노드로 추가**
  - 탐색이 성공한 경우 → 데이터가 이미 존재하므로 삽입 없이 종료



# 삭제 연산

## 후속자 노드 successor, 계승자 노드

- 어떤 노드의 키값 바로 다음 키값을 갖는 노드




7 15 22 30 35 40 44 55 88

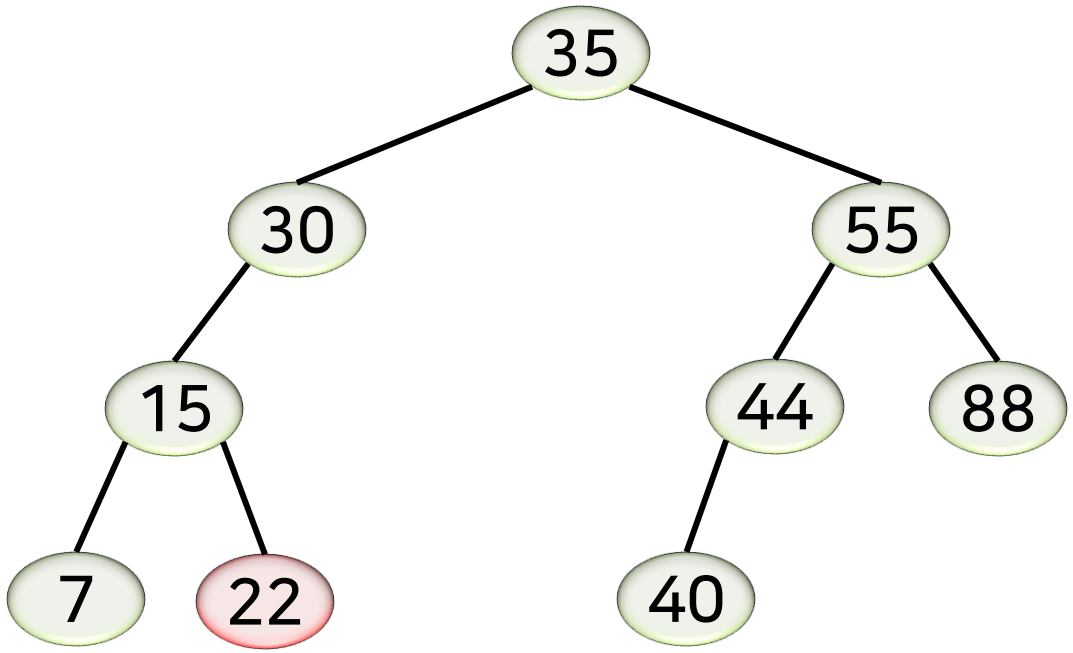
# 삭제 연산

## 삭제되는 노드의 자식 노드의 개수에 따라 구분해서 처리

1. 자식 노드가 없는 경우(리프 노드의 경우)
   - 남는 노드가 없으므로 위치 조절이 불필요
2. 자식 노드가 1개인 경우
   - 자식 노드를 삭제되는 노드의 위치로 올리면서 서브트리 전체도 따라 올림
3. 자식 노드가 2개인 경우
   - 삭제되는 노드의 후속자 노드를 삭제되는 노드의 위치로 올리고,
   - 후속자 노드를 삭제되는 노드로 취급하여 자식 노드의 개수에 따라 다시 처리

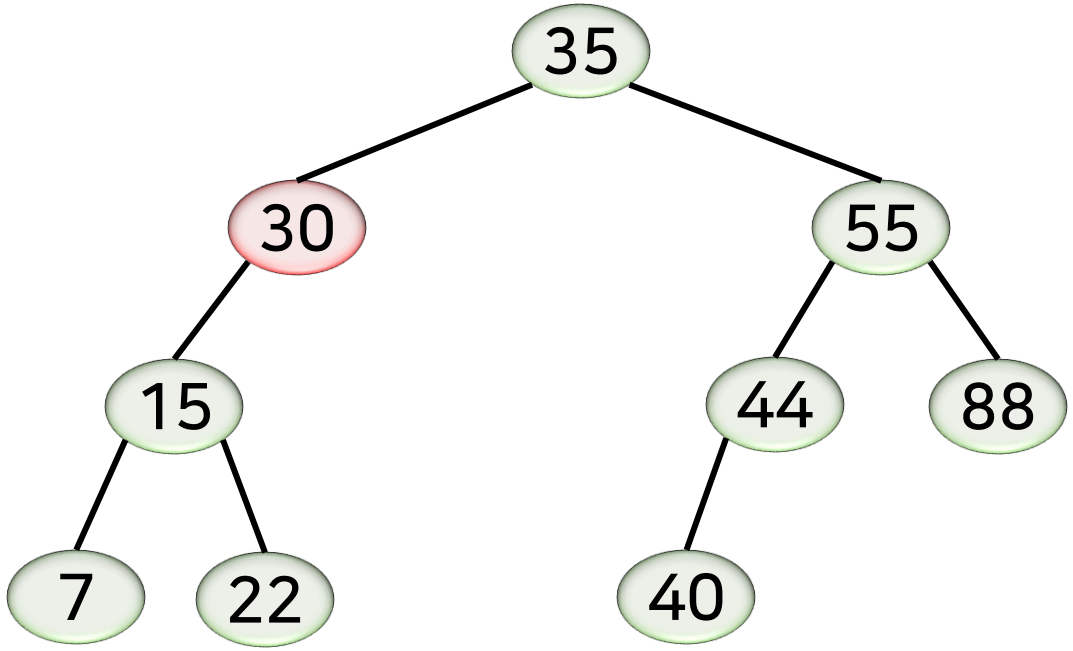
# 삭제 연산

## 노드 22 삭제



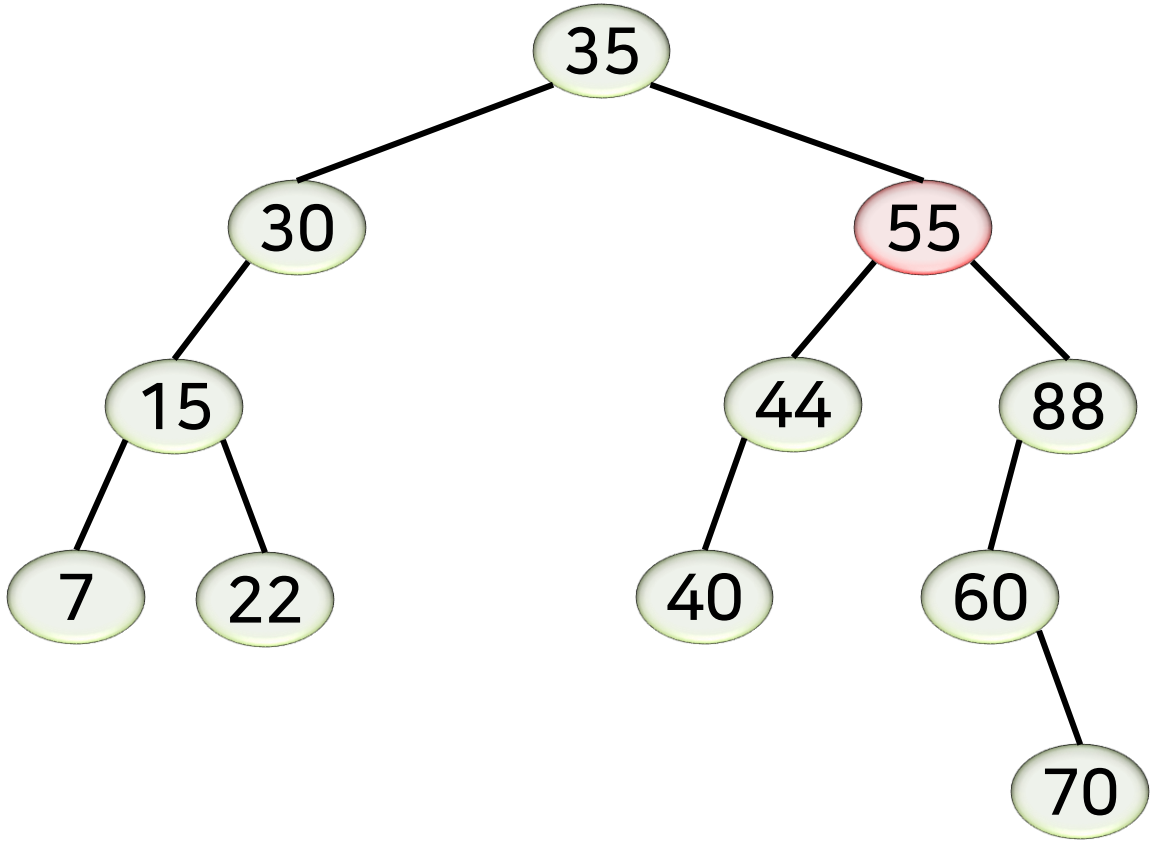
# 삭제 연산

## 노드 30 삭제



# 삭제 연산

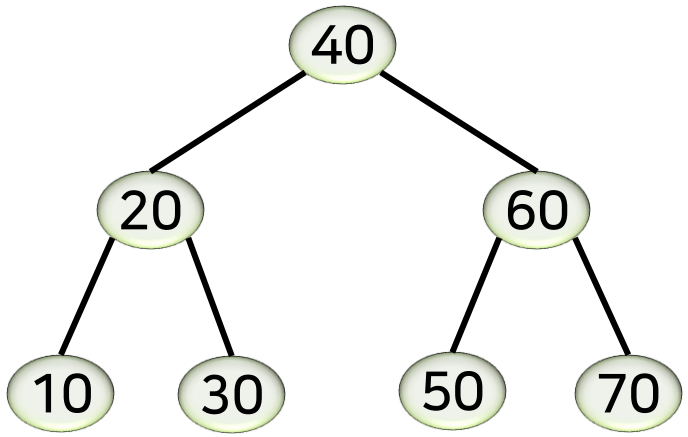
## 노드 55 삭제



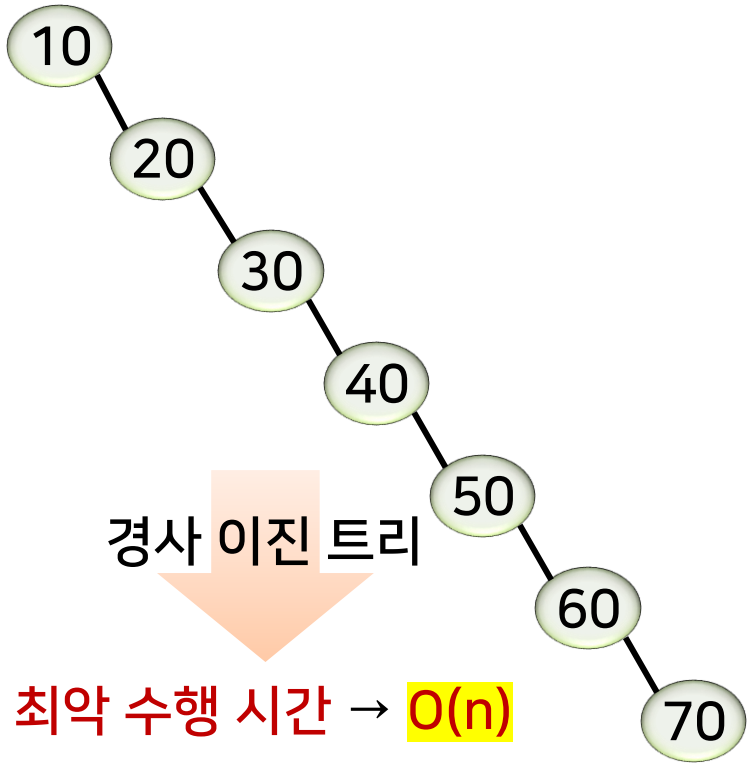
# 성능

## 키값의 비교 횟수에 비례

- 트리의 높이 h → O(h)




평균 수행 시간 → O(logn)




경사 이진 트리

최악 수행 시간 → O(n)

# 정리하기

## 1. 정렬 알고리즘: 퀵 정렬, 합병 정렬

- 퀵 정렬 → 분할정복 방법, 피벗, 분할 과정, $O(n\log n)$, $O(n^2)$
- 합병 정렬 → 분할정복 방법, 합병 과정, $O(n\log n)$

## 2. 순차 탐색, 이진 탐색

- 순차탐색 → 비정렬 데이터에 탐색에 적합, 데이터가 큰 경우 부적합, $O(n)$
- 이진탐색 → 정렬된 입력 배열에 대해서만 적용 가능,
  삽입/삭제가 빈번한 응용은 부적합, $O(\log n)$

## 3. 이진 탐색 트리

- 두 가지 성질을 만족하는 이진 트리
- 연산 → 탐색, 삽입, 삭제(후속자 노드, 자식 노드의 개수에 따라 구분)
- 최선/평균 탐색 시간 $O(\log n)$, 최악 탐색 시간 $O(n)$

07 강

다음시간 안내

# 운영체제 (1)

## 08강. 운영체제 (2)

09 강 **컴퓨터 구조 (1)**