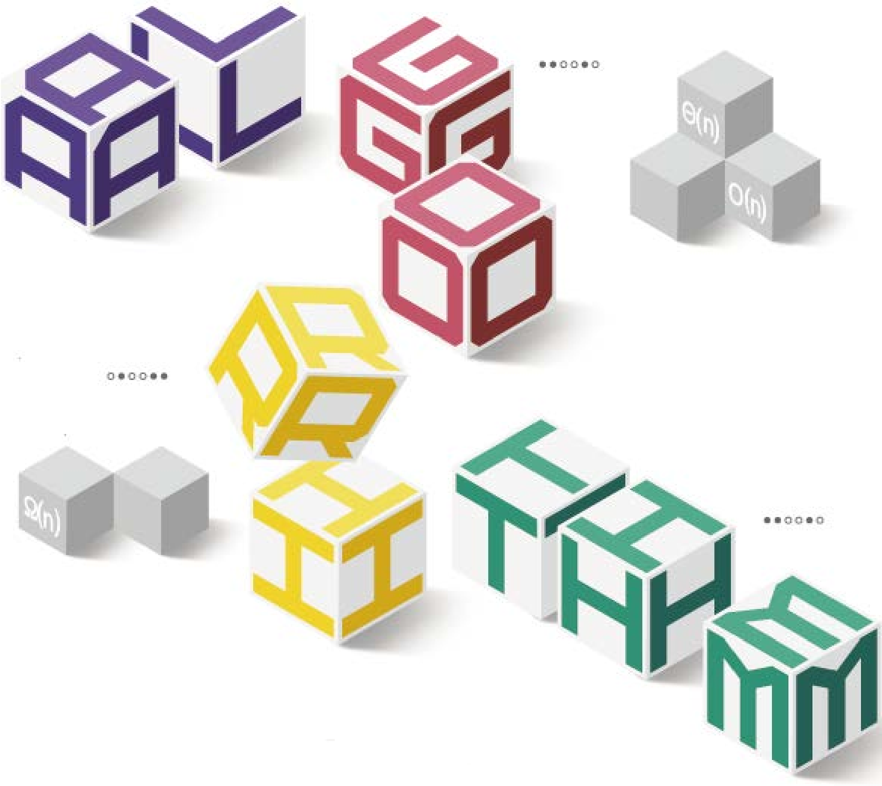


관계 중심의 사고법

# 쉽게 배우는 알고리즘

## 4장. 정 렬 Sorting

# 4장. 정렬Sorting

# 학습목표

- 정렬 알고리즘을 수행 시간에 의해 분류한다.
- 비교 정렬의 한계를 이해하고, 선형 시간 정렬이 가능한 조건과 선형 시간 정렬 알고리즘을 이해한다.

# 정렬Sorting 알고리즘들

- 대부분 $O(n^2)$과 $O(n\log n)$ 사이
- Input이 특수한 성질을 만족하는 경우에는 $O(n)$ sorting도 가능
  - E.g., input이 $-O(n)$과 $O(n)$ 사이의 정수

# 기초적인 정렬 알고리즘

- 평균적으로 $\Theta(n^2)$의 시간이 소요되는 정렬 알고리즘들
  - 선택정렬
  - 버블정렬
  - 삽입정렬

# 고급 정렬 알고리즘

- 평균적으로 $\Theta(n\log n)$의 시간이 소요되는 정렬 알고리즘들
  - 퀵정렬
  - 병합정렬
  - 힙정렬

# 퀵정렬

```
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;
}
```

# Animation (퀵정렬)

1 2 3 4 5 6 8 9

✓ 평균 수행 시간: $\Theta(n\log n)$
✓ 최악의 경우 수행 시간: $\Theta(n^2)$

# 퀵정렬의 작동 예

정렬할 배열이 주어짐. 첫번째 수를 기준으로 삼는다.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 31 | 8 | 48 | 73 | 11 | 3 | 20 | 29 | 65 | 15 |

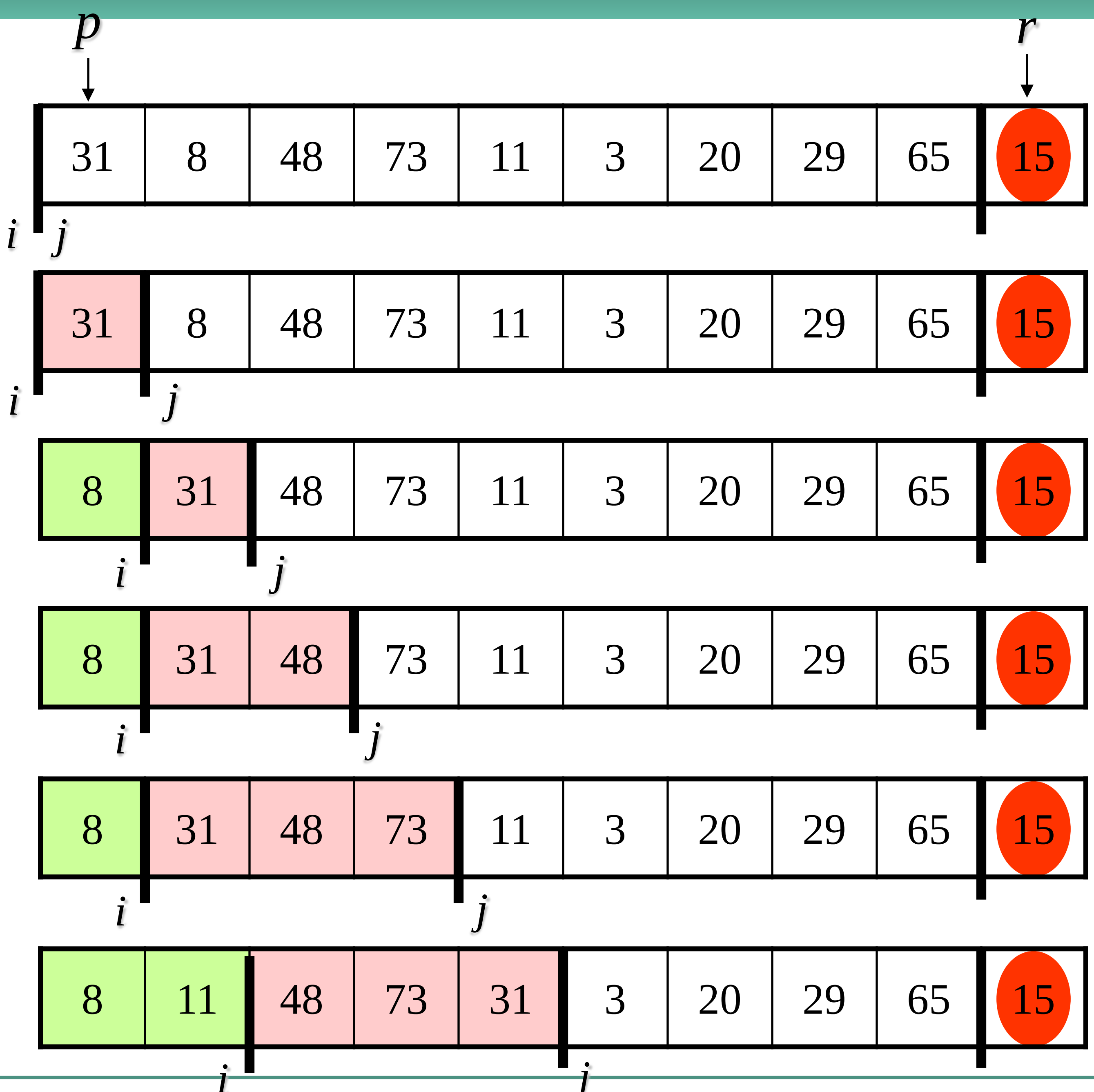
기준(15)보다 작은 수는 기준의 왼쪽에, 나머지는 기준의 오른쪽에 오도록 재배치한다

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 11 | 3 | 15 | 31 | 48 | 20 | 29 | 65 | 73 | (a) |

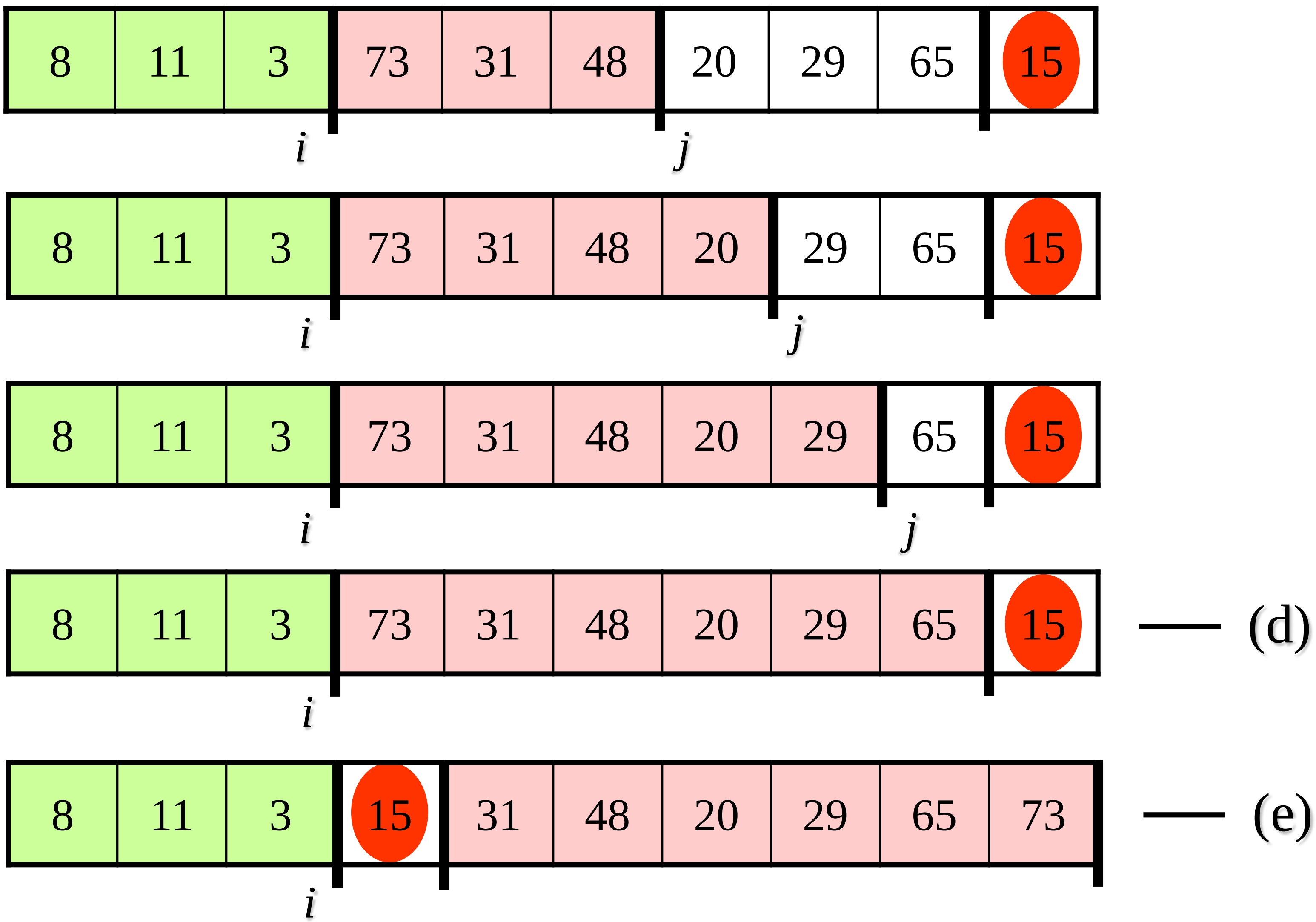
기준원소(15) 왼쪽과 오른쪽을 독립적으로 정렬한다 (정렬완료)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 8 | 11 | 15 | 20 | 29 | 31 | 48 | 65 | 73 | (b) |

# 분할 (partition)



# 분할 (partition)



# $\Theta(n)$ 정렬

- 두 원소를 비교하는 것을 기본 연산으로 하는 정렬의 하한선은 $\Omega(n\log n)$이다
- 그러나 원소들이 특수한 성질을 만족하면 $\Theta(n)$ 정렬도 가능하다
  - 계수정렬Counting Sort
    - 원소들의 크기가 모두 $-O(n) \sim O(n)$ 범위에 있을 때
  - 기수정렬Radix Sort
    - 원소들이 모두 $k$ 이하의 자릿수를 가졌을 때 ($k$: 상수)

# Lower bound for sorting

- Comparison sort: 두 원소의 비교만을 사용하는 알고리즘
- Decision tree: 정렬 알고리즘의 비교 과정을 보여주는 binary tree (insertion sort, n=3)
  - internal node: 비교 (i:j)
  - leaf: 입력 숫자들의 순서를 나타내는 permutation
  - 특정 입력에 대한 정렬 알고리즘의 수행은 decision tree의 root에서 leaf까지의 path에 해당된다.
  - 정렬 알고리즘이 맞다면, n!개의 permutation이 모두 decision tree의 leaf에 나타나야 한다.

## Lower bound

- Decision tree의 root에서 leaf까지의 longest path (tree의 높이)가 정렬 알고리즘의 worst case이다.
- Decision tree의 높이가 h, leaf의 개수가 f라고 하면
  - $n! \leq f \leq 2^h$
  - $h \geq \log n!$ 따라서 $h = \Omega(n \log n)$

# 계수정렬 Counting Sort

```
countingSort(A, B, n)
▷ A[1…n]: 입력 배열
▷ B[1…n]: 배열 A를 정렬한 결과
{
        for i = 1 to k
                C[i] ← 0;
        for j = 1 to n
                C[A[j]]++;
        ▷ 이 시점에서의 C[i] : 값이 i인 원소의 총 수
        for i = 1 to k
                C[i] ← C[i] + C[i-1] ;
        ▷ 이 시점에서의 C[i] : i보다 작거나 같은 원소의 총 개수
        for j ← n downto 1 {
                B[C[A[j]] ← A[j];
                C[A[j]]--;
        }
}
```

# 기수정렬 Radix Sort

radixSort(A[ ], $n$, $d$)

▷ 원소들이 각각 최대 $d$ 자리수인 A[1…$n$]을 정렬한다

▷ 가장 낮은 자리수를 1번째 자리수라 한다

{

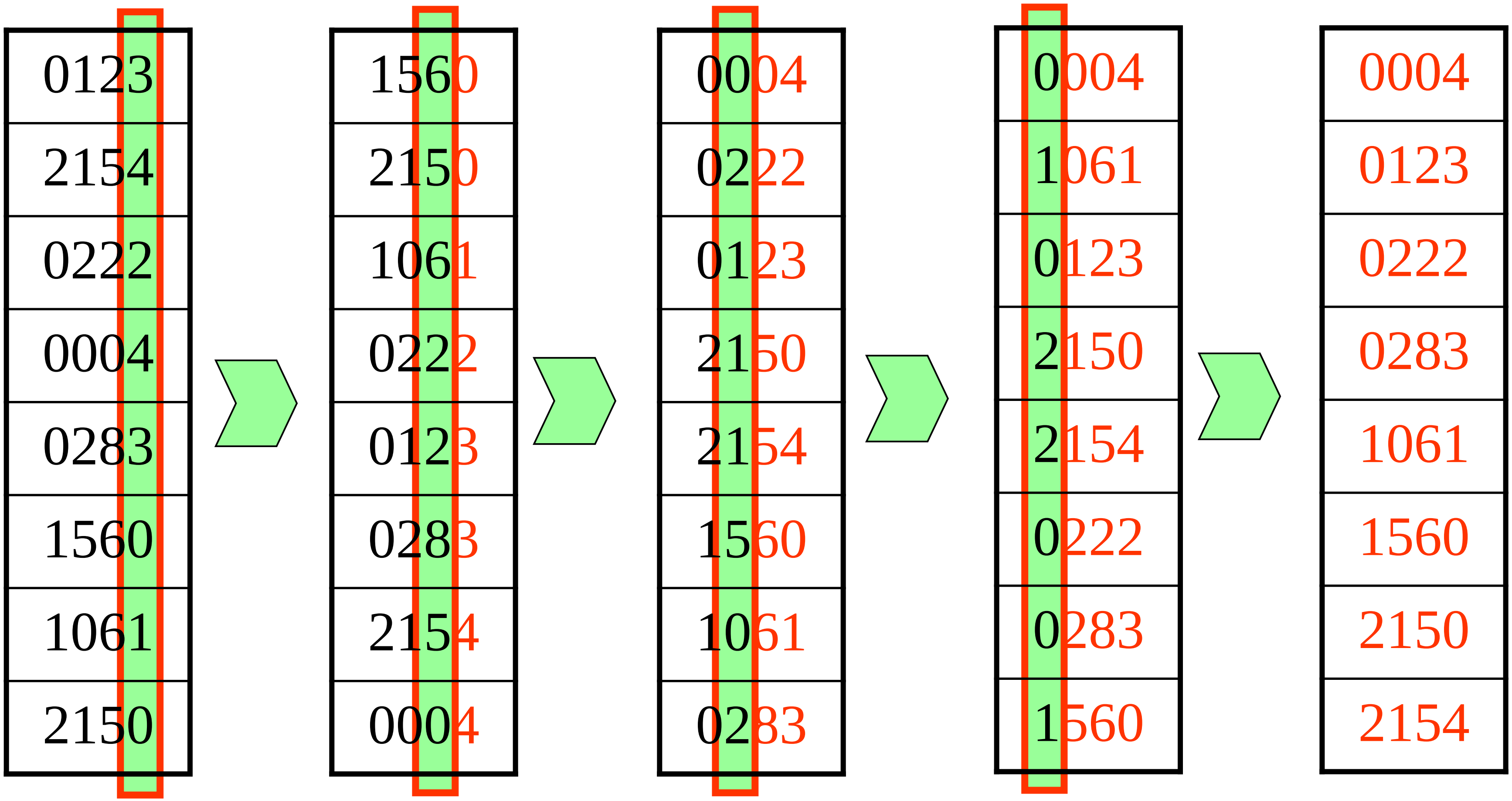
**for** $i \leftarrow 1$ to $d$

$i$ 번째 자리수에 대해 A[1…$n$] 을 안정을 유지하면서 정렬한다;

}

✓ 안정성 정렬 Stable sort

— 같은 값을 가진 원소들은 정렬 후에도 원래의 순서가 유지되는 성질을 가진 정렬을 일컫는다.




✓ Running time: $\Theta(n)$ ← $d$: a constant

# 효율성 비교

| | **Worst Case** | **Average Case** |
|---|---|---|
| Selection Sort | $n^2$ | $n^2$ |
| Bubble Sort | $n^2$ | $n^2$ |
| Insertion Sort | $n^2$ | $n^2$ |
| Quicksort | $n^2$ | $n\log n$ |
| Mergesort | $n\log n$ | $n\log n$ |
| Heapsort | $n\log n$ | $n\log n$ |
| Counting Sort | $n$ | $n$ |
| Radix Sort | $n$ | $n$ |

# Thank you