

분할정복 방법_1

2020년도 2학기 최 희 석

목차

▶ 분할정복 방법

▶ 대표적 분할정복 방법

▶ 정렬

▶ 퀵 정렬

▶ 합병 정렬

설계 기법

▶ 순차탐색 (미 정렬) vs 이진탐색 (정렬)

▶ 문제와 제반 조건이 매우 다양

▶ 일반적인 기법은 없음

▶ 대표적인 설계 기법

▶ 분할정복 방법

▶ 욕심쟁이 방법

▶ 동적 프로그래밍 방법

분할정보 방법

분할정복 방법의 원리

순환적recursive으로 문제를 푸는 하향식 접근 방법

- ✓ 문제를 더 이상 나눌 수 없을 때까지 작은 문제로 나누고, 이렇게 나누어진 문제들을 각각 해결한 후 이들의 해를 결합하여 원래 문제의 해를 구하는 하향식top-down 접근 방법

분할 주어진 문제를 여러 개의 작은 문제로 분할

정복 작은 문제들을 순환적으로 분할. 만약 작은 문제가 더 이상 분할되지 않을 정도로 크기가 충분히 작다면 순환 호출 없이 작은 문제에 대한 해를 구함

결합 작은 문제에 대해 정복된 해를 결합하여 원래 문제의 해를 구함

분할정복 방법의 원리

특징

✓ 분할된 작은 문제는 원래의 문제의 성격과 동일

-> 단, 입력 크기만 작아짐

✓ 분할된 문제는 서로 독립적

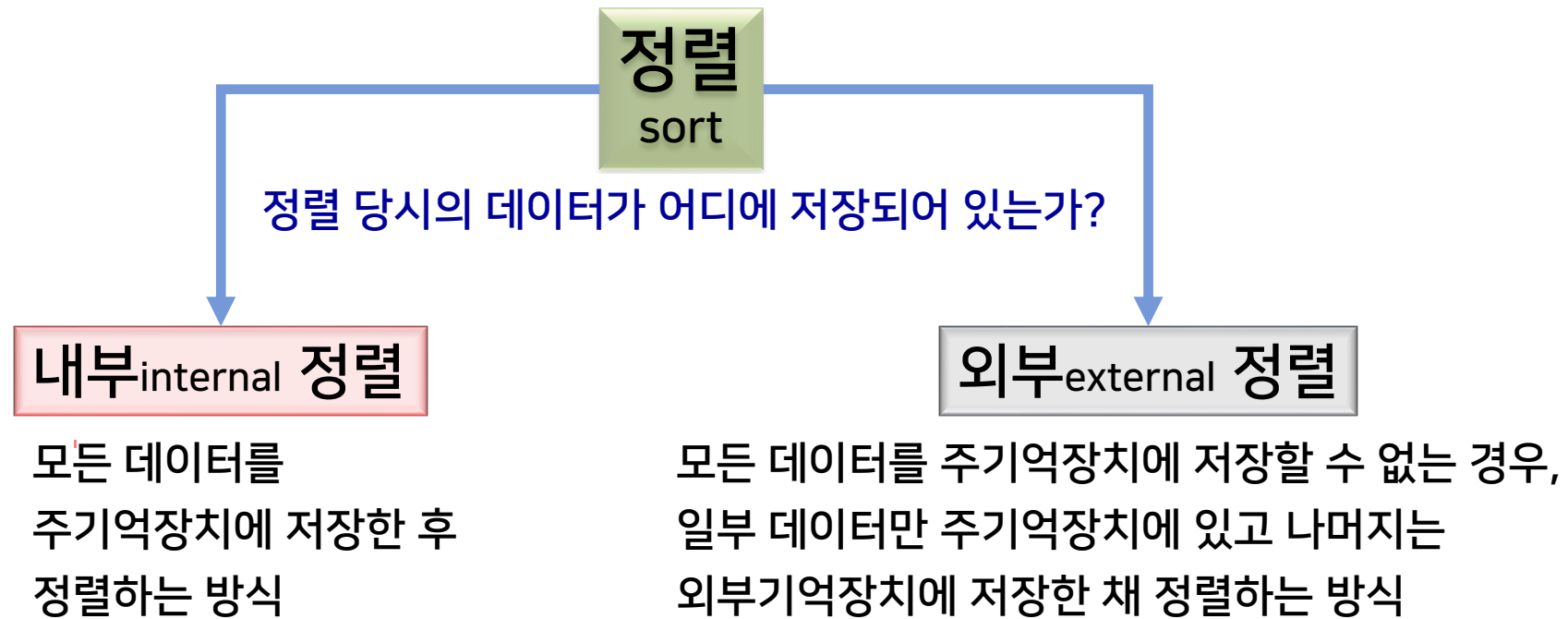
-> 순환적 분할 및 결과 결합이 가능

대표적 분할정복 방법

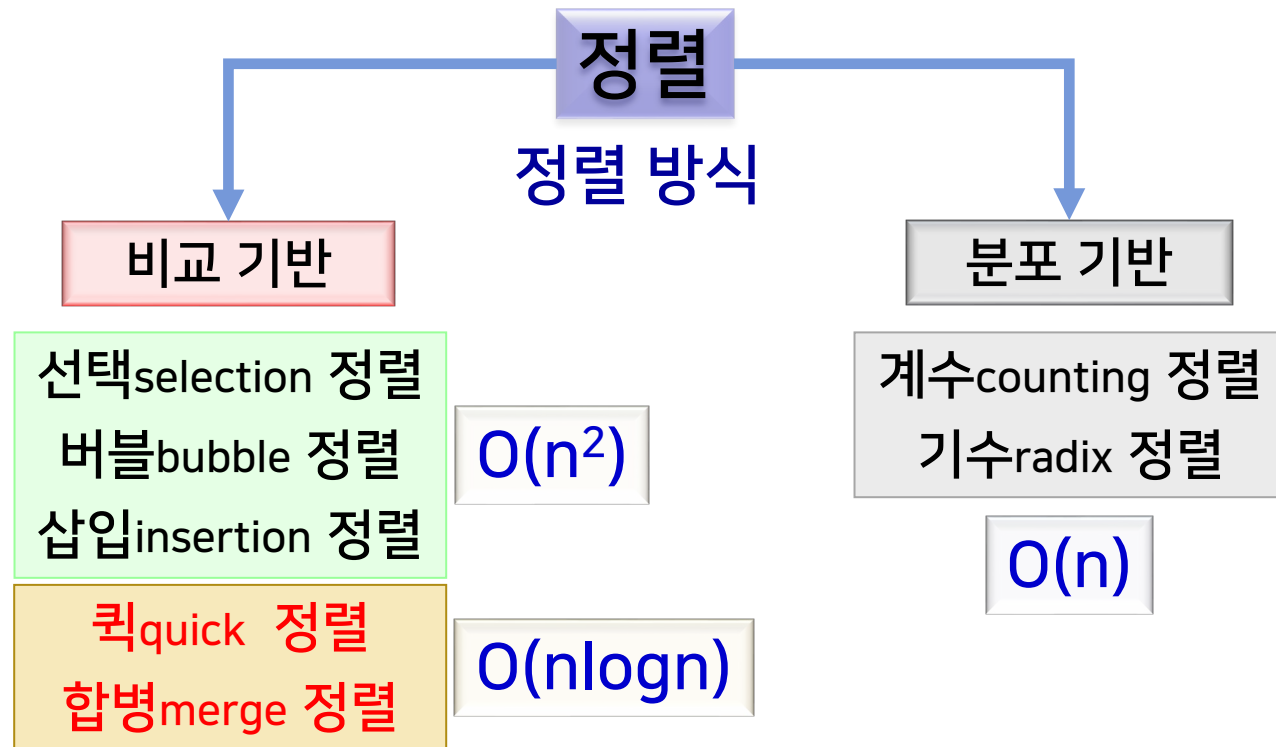
정렬

기본 개념

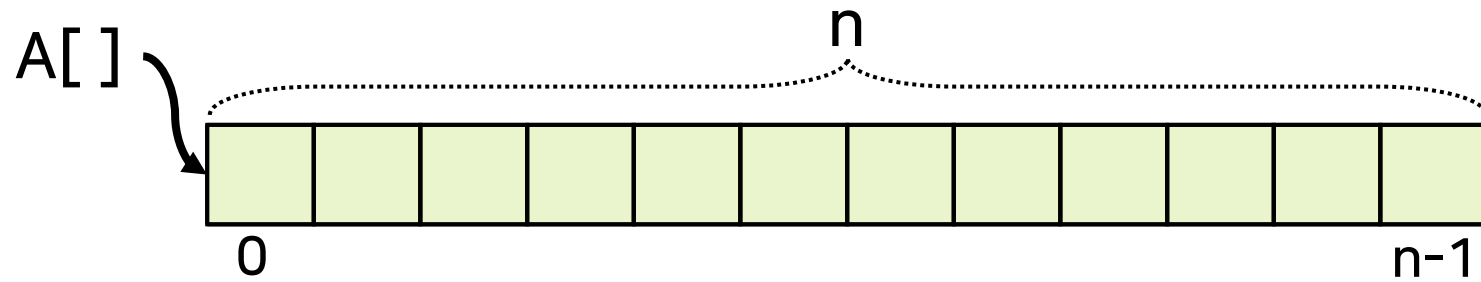
🔄 정렬 → 주어진 데이터를 값의 크기 순서에 따라 재배열하는 것



기본 개념



정렬을 위한 기본 가정



$A[i] > 0, (0 \leq i \leq n-1)$

if $i < j$ then $A[i] \leq A[j], (0 \leq i, j \leq n-1)$

- 키값 \rightarrow 양의 정수
- 정렬 방식 \rightarrow 오름차순
- 키의 개수 $\rightarrow n$
- 키 저장 $\rightarrow A[0..n-1]$

퀵 정렬

퀵 정렬

↻ 특정한 원소를 기준으로 주어진 입력 리스트의 원소를 두 개의 서브리스트로 분할하고, 각 서브리스트에 대해서 독립적으로 퀵 정렬을 순환적으로 적용하여 정렬하는 방식

✓ 평균적으로 가장 좋은 성능의 비교 기반 알고리즘 → $O(n \log n)$

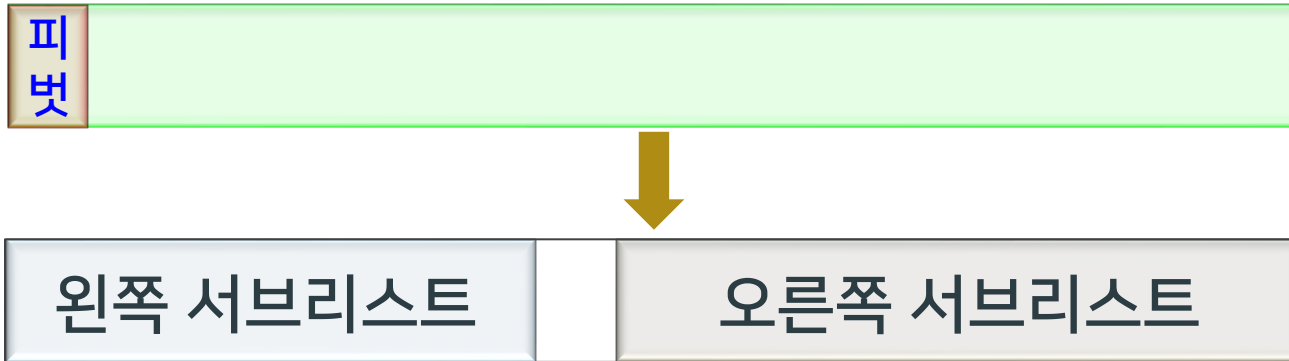
↻ 피벗 pivot

✓ 두 개의 서브리스트로 분할할 때 기준이 되는 원소("분할원소")

- 보통 주어진 리스트에서 가장 왼쪽의 원소(첫 번째 원소)로 지정

퀵 정렬의 원리

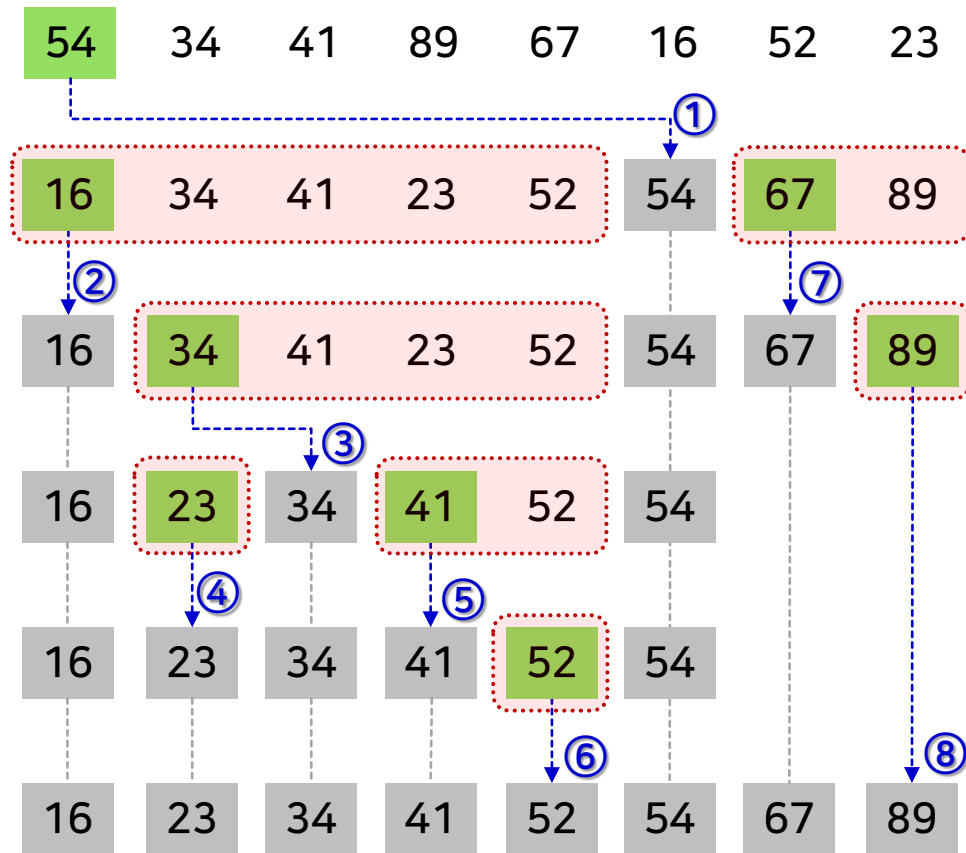
🔄 피벗이 제자리를 잡도록 정렬하는 방식



왼쪽 부분배열의 모든 값 < 피벗

피벗 < 오른쪽 부분배열의 모든 값

퀵 정렬의 전체 수행 과정



알고리즘_퀵정렬

```
QuickSort (A[ ], n)
```

```
{
```

```
    if (n>1){
```

```
        pivot = Partition(A[0..n-1], n);
```

//두 부분배열로 분할

```
        QuickSort(A[0..pivot-1], pivot);
```

//왼쪽 부분배열에 대한 순환 호출

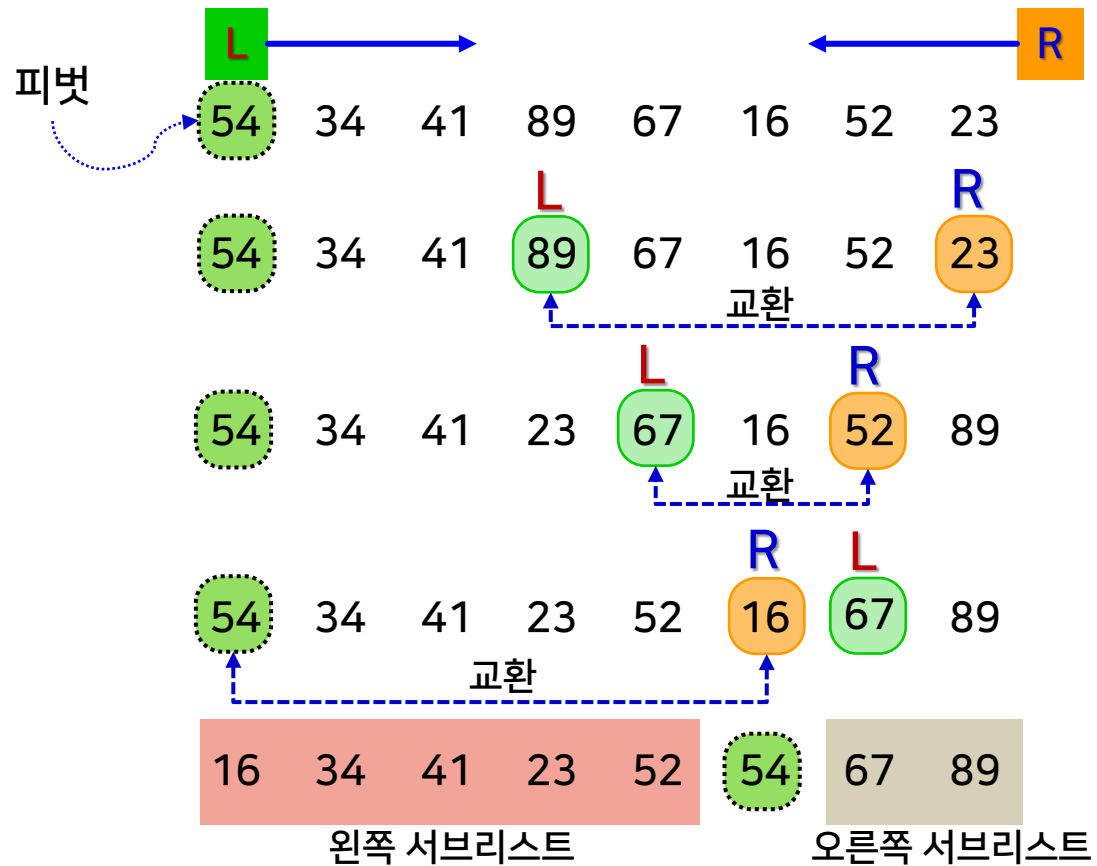
```
        QuickSort(A[pivot+1..n-1], n-pivot-1);
```

//오른쪽 부분배열에 대한 순환 호출

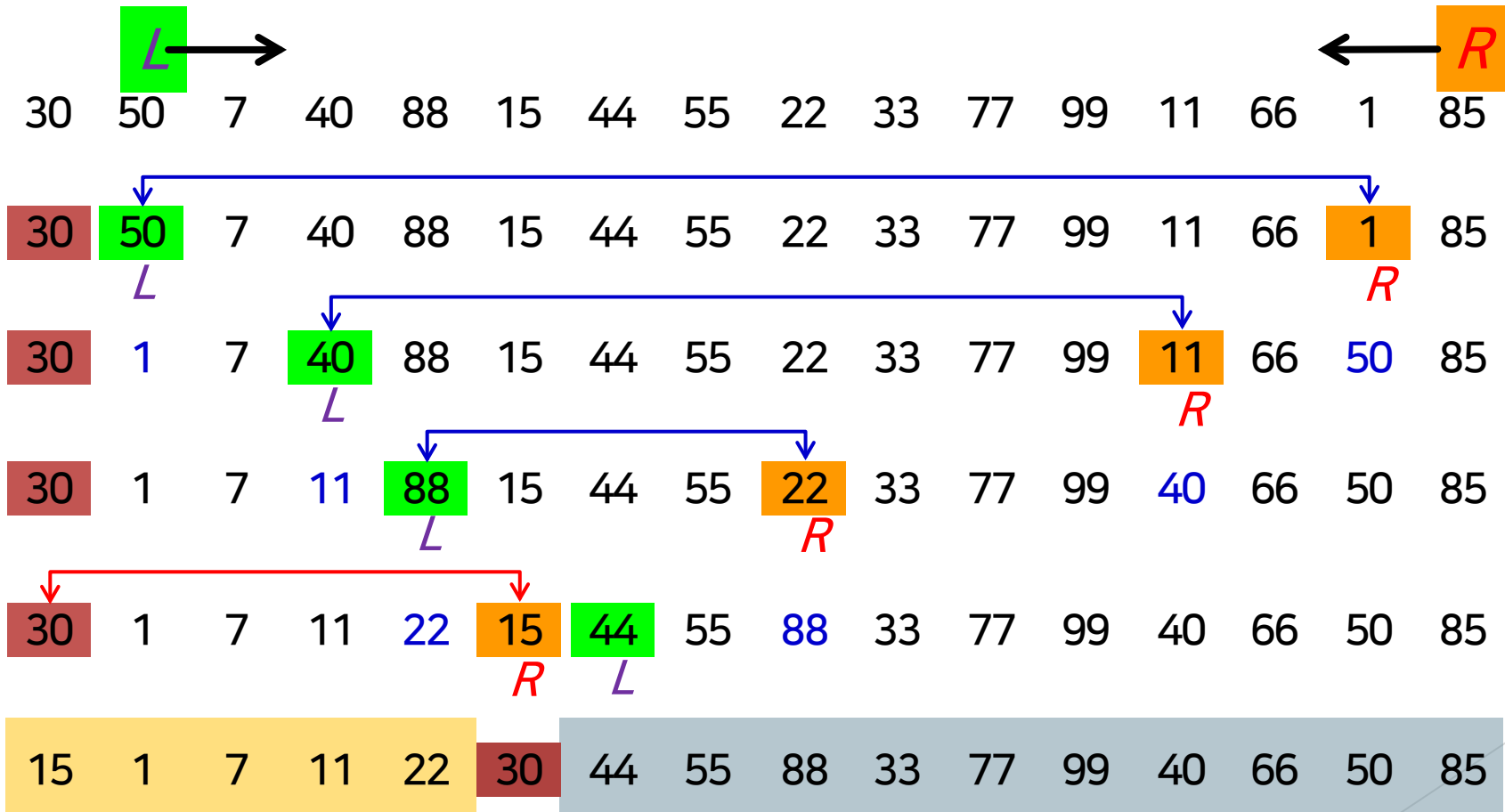
```
    }
```

```
}
```

분할 partition 과정



분할 partition 과정



알고리즘_분할과정

```
int Partition (A[ ], n)
{
    Left = 1; Right = n-1;
    while ( Left < Right ) {                // 피벗 A[0]
        //피벗보다 큰 값의 위치를 찾음
        while ( Left < n && A[Left] < A[0] ) Left++;
        //피벗보다 작은 값의 위치를 찾음
        while ( Right > 0 && A[Right] >= A[0] ) Right--;
        if ( Left < Right ) 교환(A[Left]  $\leftrightarrow$  A[Right])
        else 교환(A[0]  $\leftrightarrow$  A[Right])
    }
    return Right;
}
```

퀵 정렬의 특징

🔄 분할정복 방법을 적용한 알고리즘

✓ 분할

- 정렬 할 n 개의 원소를 피벗을 중심으로 두 개의 서브리스트로 분할

✓ 정복

- 두 서브리스트에 대해 퀵 정렬을 각각 순환적으로 적용하여 두 서브리스트를 정렬

✓ 결합

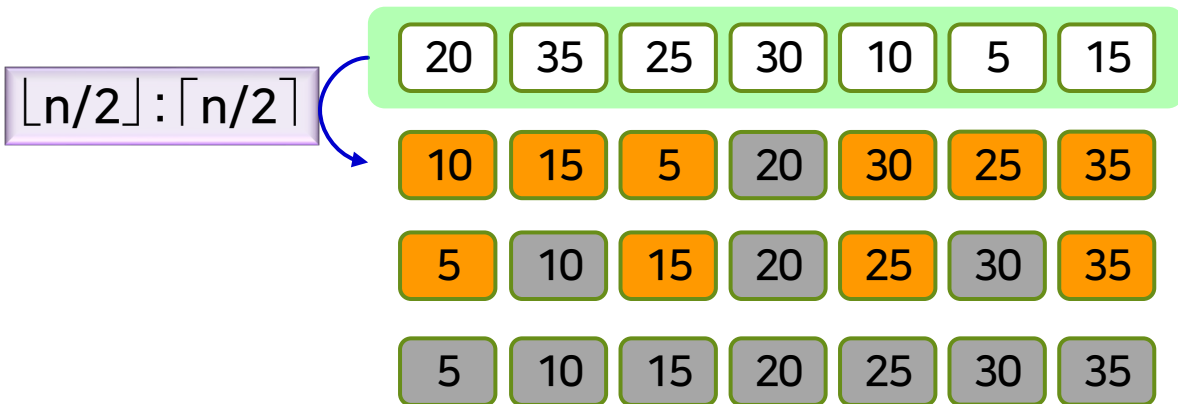
- 필요 없음

퀵 정렬의 성능 (1/2)

🔄 분할 과정(분할 함수)의 수행시간 → $O(n)$

🔄 최선 수행 시간 → $O(n \log n)$

✓ 피벗을 중심으로 동일한 크기의 두 개의 서브리스트로 분할되는 경우



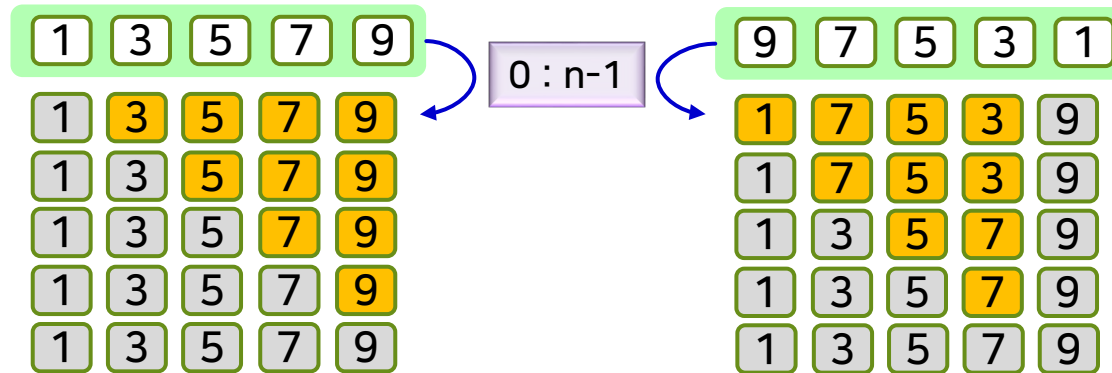
시간 복잡도

- ▶ 알고리즘의 수행시간
 - ▶ 실제 수행되는 시간을 측정하지 않음 → 컴퓨터 및 프로그래밍 언어 등에 의존
 - ▶ 알고리즘에 사용된 단위 연산들의 수행 횟수의 합
- ▶ 입력의 크기
 - ▶ 단순히 단위 연산의 개수가 아닌 **입력 크기의 함수**로 표현
- ▶ **입력 데이터의 상태**에 따라 달라짐
 - ▶ 평균 수행 시간, 최선 수행 시간, **최악 수행 시간**

퀵 정렬의 성능 (2/2)

🔄 최악 수행 시간 → $O(n^2)$

- ✓ 피벗 하나만 제자리를 잡고 나머지 모든 원소가 하나의 서브리스트로 분할되는 경우



- 피벗이 리스트에서 항상 최대값 또는 최소값이 되는 경우
- 입력 데이터가 이미 정렬되고 **AND** 피벗이 맨 왼쪽 원소로 선택되는 경우

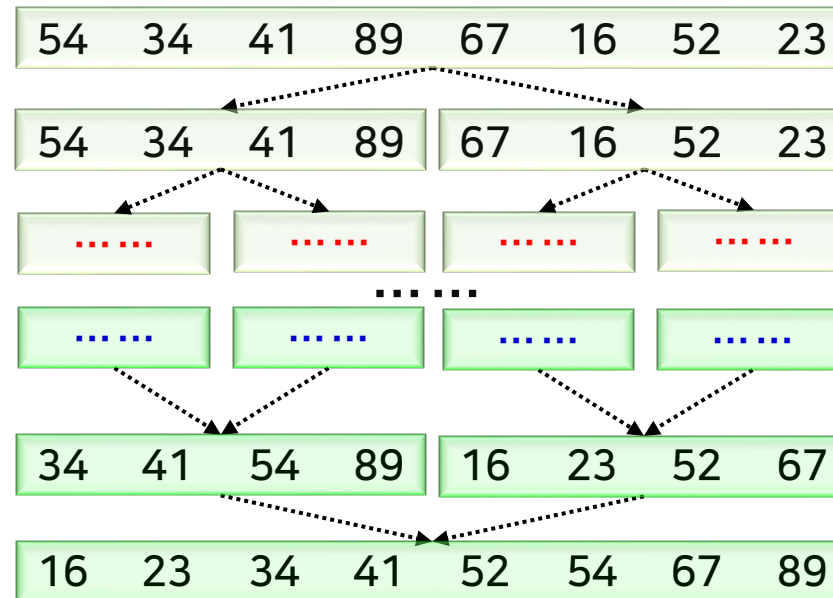
→ 피벗 선택의 임의성만 확보되면 평균 수행 시간이 보장됨

- 최악수행 막기: 배열에서 임의로 값을 선택해서 배열의 처음 원소와 서로 교환한 후 정렬 수행

합병 정렬

합병 정렬

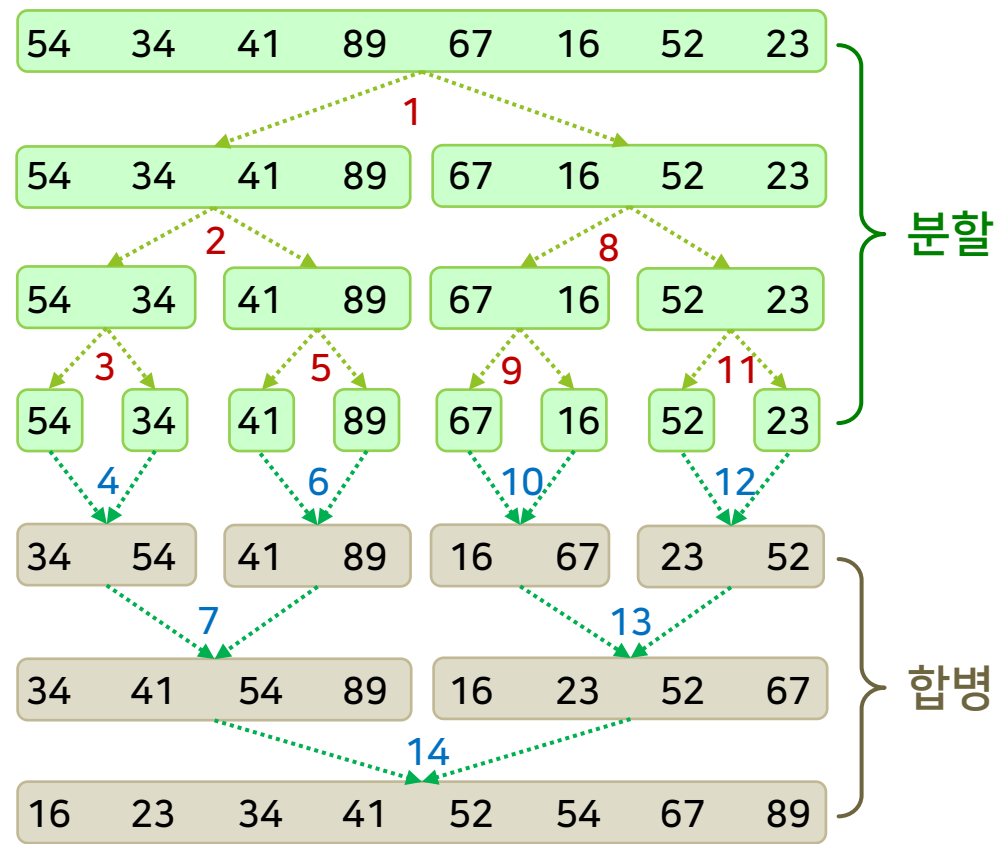
- 동일한 크기의 두 개의 부분배열로 분할하고, 각 부분배열을 순환적으로 정렬한 후, 정렬된 두 부분배열을 합병하여 하나의 정렬된 배열을 형성하는 방식



알고리즘_합병 정렬

```
MergeSort (A[ ], n)
{
    if (n > 1) {
        Mid = ⌊n/2⌋;
        B[0..Mid-1] = MergeSort(A[0..Mid-1], Mid);
        C[Mid..n-1] = MergeSort(A[Mid..n-1], n-Mid);
        A[0..n-1] = Merge(B[0..Mid-1], C[Mid..n-1], Mid, n-Mid);
    }
    return A;
}
```


합병 정렬의 전체 수행 과정

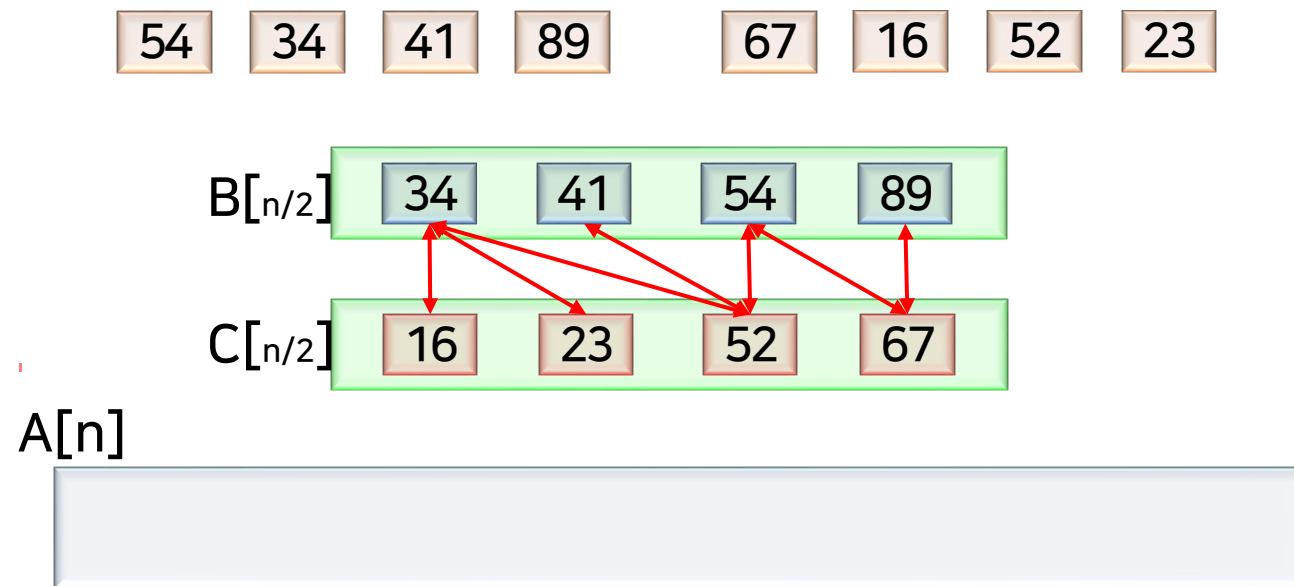


알고리즘_합병 함수 Merge()

```
Merge (B[ ], C[ ], n, m)
{
    i = j = k = 0;
    while ( i < n && j < m ) {
        if ( B[i] <= C[j] )
            A[k++] = B[i++];
        else
            A[k++] = C[j++];
    }
    for ( ; i < n; i++ ) A[k++] = B[i];
    for ( ; j < m; j++ ) A[k++] = C[j];
    return A[0..n+m-1];
}
```

합병 정렬

🔄 합병 merge 과정



합병 정렬

	B[i]				C[j]				A[k] ← B[i]+C[j]							
	0	1	2	3	0	1	2	3	0	1	2	3	4	5	6	7
1	15	20	35	50	10	25	30	40	10							
2	15	20	35	50	10	25	30	40	10	15						
3	15	20	35	50	10	25	30	40	10	15	20					
4	15	20	35	50	10	25	30	40	10	15	20	25				
5	15	20	35	50	10	25	30	40	10	15	20	25	30			
6	15	20	35	50	10	25	30	40	10	15	20	25	30	35		
7	15	20	35	50	10	25	30	40	10	15	20	25	30	35	40	
8	15	20	35	50	10	25	30	40	10	15	20	25	30	35	40	50

합병 정렬의 특징

🔄 분할정복 방법을 적용한 알고리즘

✓ 분할

- 정렬할 n 개의 원소를 $n/2$ 개의 원소를 갖는 두 개의 서브리스트로 분할

✓ 정복

- 두 서브리스트에 대해 분할 정렬을 각각 순환적으로 적용하여 두 서브리스트를 정렬

✓ 결합

- 정렬된 두 서브리스트를 합병하여 하나의 정렬된 리스트를 생성

🔄 최선, 최악, 평균 수행 시간 → $O(n \log n)$

과제 안내

퀵 정렬

QuickSort(A, left, right)

입력: 배열 A[left]~A[right]

출력: 정렬된 배열 A[left]~A[right]

If (left < right) {


 피벗을 A[left]~A[right] 중에서 선택하고, 피벗을 A[left]와 자리를 바꾼 후,
 피벗과 배열의 각 원소를 비교하여 피벗보다 작은 숫자들은 A[left]~A[p-1]로 옮
 기고, 피벗보다 큰 숫자들은 A[p+1]~A[right]로 옮기며, 피벗은 A[p]에 놓는다.

 QuickSort(A, left, p-1)

 QuickSort(A, p+1, right)

}

퀵 정렬

 Microsoft Visual Studio 디버그 콘솔

```
Enter number of elements in the array:  
3  
Enter 3 integers  
22  
11  
44  
Sorted array:  
11 22 44
```


합병 정렬

MergeSort(A, p, q)

입력: $A[p] \sim A[q]$

출력: 정렬된 $A[p] \sim A[q]$

If ($p < q$) {

$k = \lfloor (p + q) / 2 \rfloor$

 MergeSort(A, p, k)

 MergeSort(A, k+1, q)

$A[p] \sim A[k]$ 와 $A[k+1] \sim A[q]$ 를 합병한다.

}

합병 정렬

Microsoft Visual Studio 디버그 콘솔

```
Enter number of elements in the array:  
10  
Enter 10 integers  
45  
2  
67  
22  
39  
553  
1  
8  
42  
31  
Printing the sorted array:  
1  
2  
8  
22  
31  
39  
42  
45  
67  
553
```

2주차 과제

- ▶ **퀴 정렬, 합병정렬 구현하기**
 - ▶ e-Class 업로드
- ▶ 양식 (한글, 워드, PDF -> 자유)
- ▶ 파일명 (이름_학번_전공)
 - ▶ 예) 최희석_2014182009_게임공학

- ▶ 질의 응답은 e-Class 질의응답 게시판에 남겨 주시길 바랍니다.