4강

분할정복 알고리즘 (2)

컴퓨터과학과 이 관 용 교수

학습목차

01 합병 정렬

02 선택 문제

학습목표

1 합병 정렬의 개념, 동작, 합병 함수, 성능, 특징을 이해하고 적용할 수 있다.

2 선택 문제의 개념과 두 종류의 알고리즘 (분할 함수 이용, 중간값들의 중간값 이용)의 원리, 동작, 성능을 이해하고 설명할 수 있다.

01

합병 정렬

[복습] 분할정복 방법

순환적으로 문제를 푸는 하향식 접근 방법

- 주어진 문제의 입력을 더 이상 나눌 수 없을 때까지 두 개 이상의 작은 문제로 순환적으로 분할하고, 이렇게 분할된 작은 문제들을 각각 해결한 후 이 해들을 결합해서 원래 문제의 해를 구하는 방식
- **'분할' – '정복' – '결합'**

특징

- 분할된 문제 → 원래 문제와 동일(입력 크기만 감소), 서로 독립적

적용 알고리즘

- 이진 탐색, 퀵 정렬, <u>합병 정렬</u>, <u>선택 문제</u>

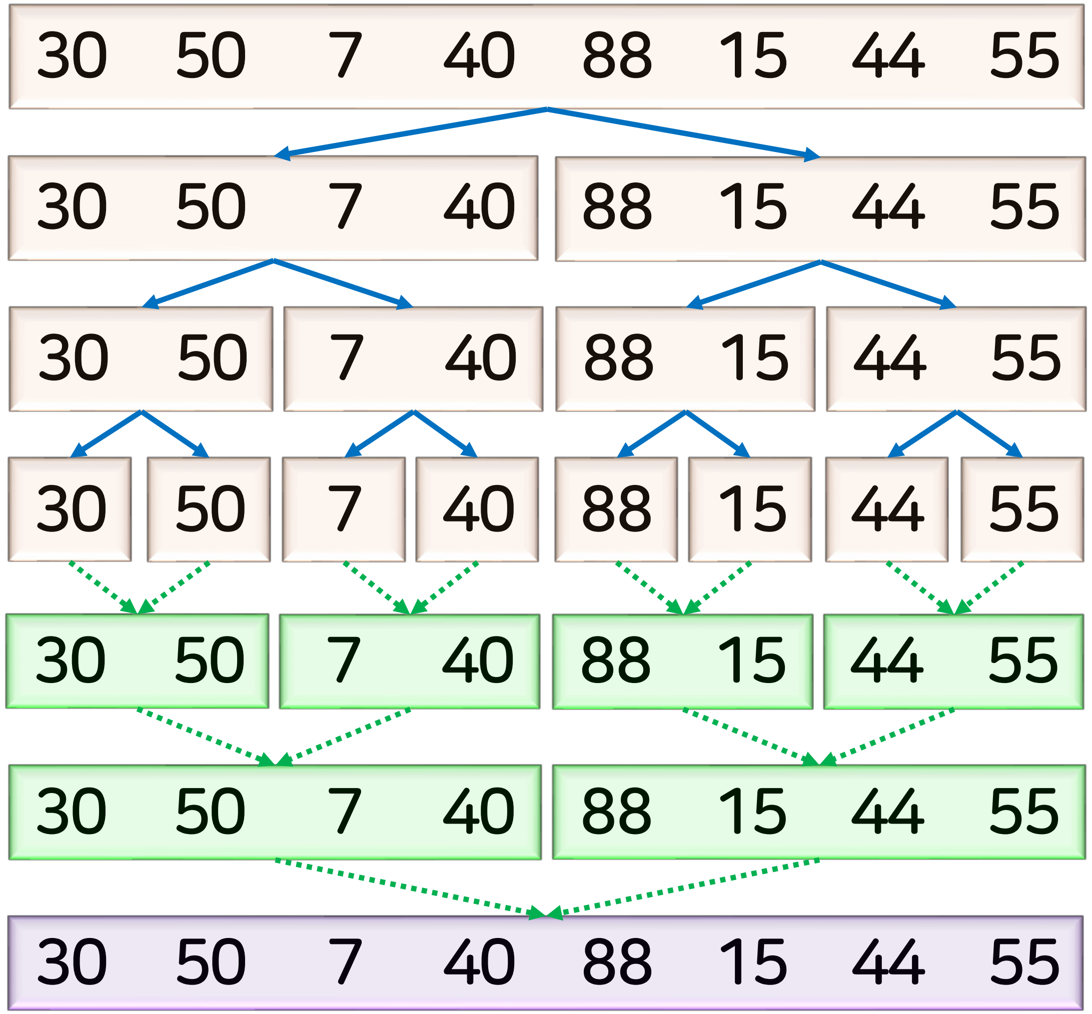
개념과 원리

전형적인 분할정복 방법이 적용된 알고리즘

- (분할) 배열을 동일한 크기의 두 개의 부분배열로 분할하고,
- (정복) 각각의 부분배열을 순환적으로 정렬한 후,
- (결합) 정렬된 두 부분배열을 합병하여 하나의 정렬된 배열을 만듦

분할	입력 크기 n인 배열을 크기 n/2인 두 부분배열로 분할한다.
정복	각각의 부분배열에 대해서 합병 정렬을 순환적으로 적용하여 두 부분배열을 정렬한다.
결합	정렬된 두 부분배열을 합병하여 하나의 정렬된 배열을 만든다.

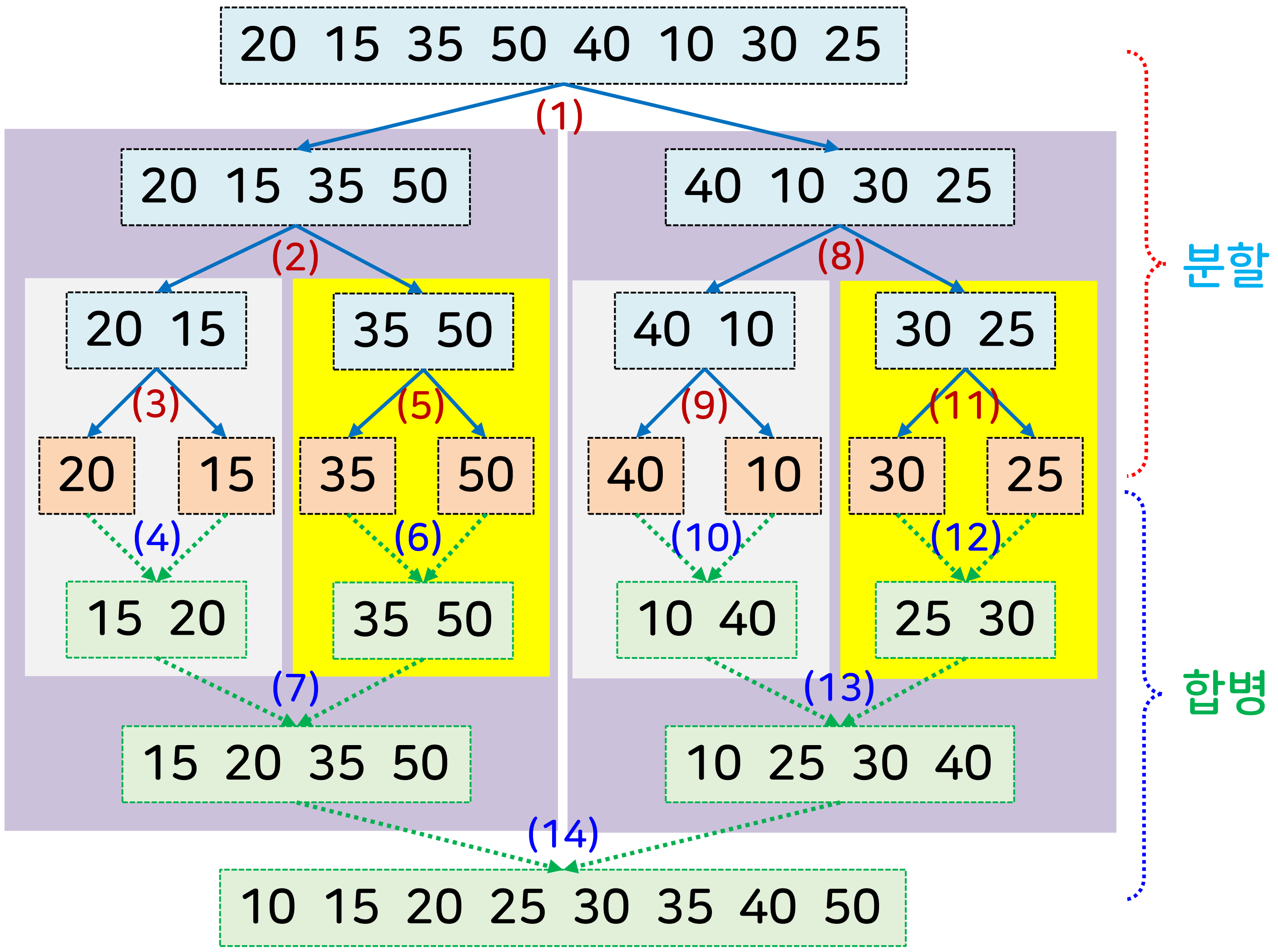
개념과 원리


30 50 7 40 88 15 44 55
30 50 7 40 | 88 15 44 55
30 50 | 7 40 | 88 15 | 44 55
30 | 50 | 7 | 40 | 88 | 15 | 44 | 55
30 50 | 7 40 | 88 15 | 44 55
30 50 7 40 | 88 15 44 55
30 50 7 40 88 15 44 55

알고리즘_합병 정렬

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

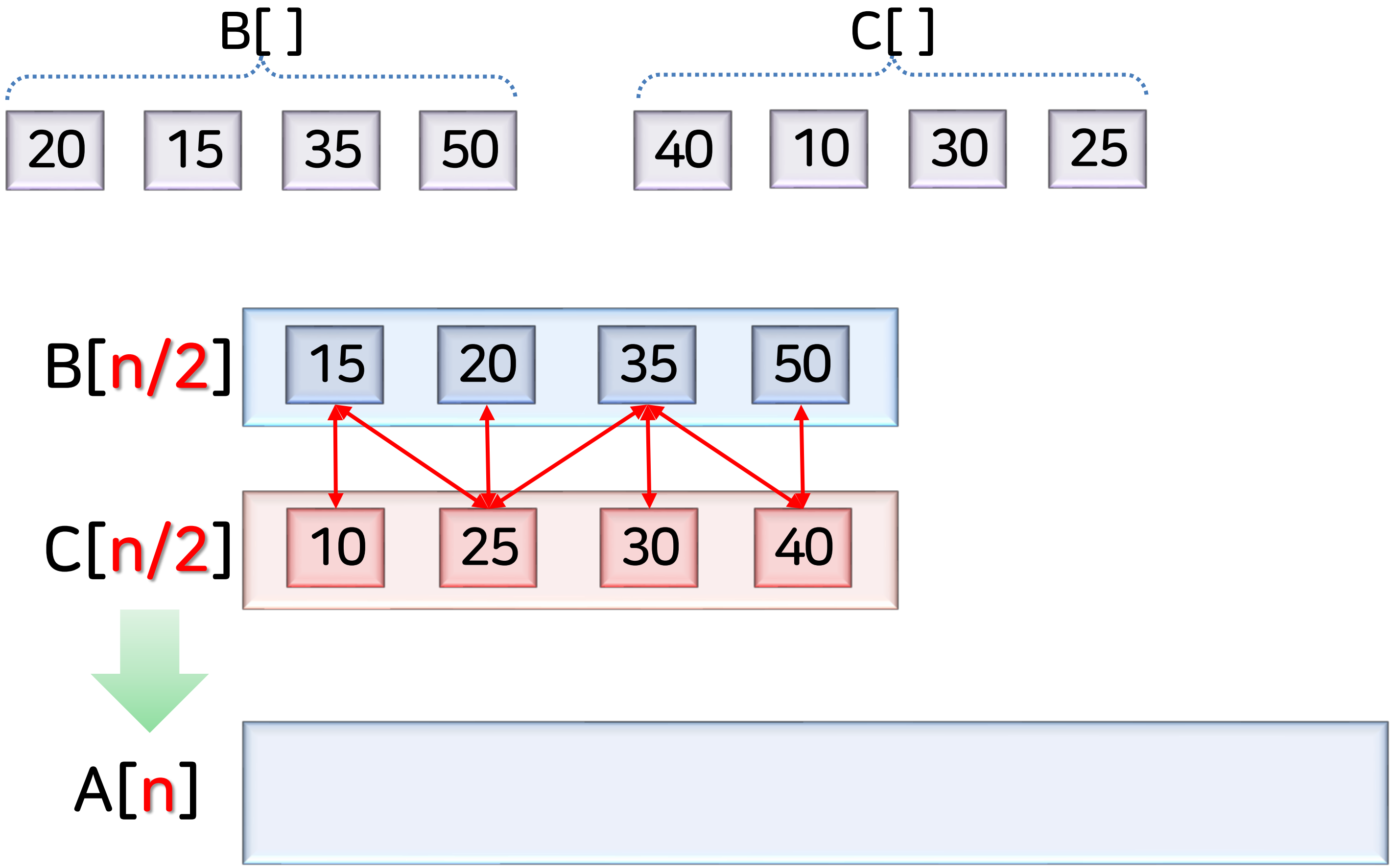
합병 정렬의 전체적인 수행 과정



알고리즘_합병 함수 Merge()

```
Merge (B[], C[], n, m)      //A[n+m-1] ← B[n] + C[m]
{
  i = j = k = 0;
  while ( i<n && j<m ) {
    if ( B[i] <= C[j] )        //비교해서 작은 값을 A[]로 이동
      A[k++] = B[i++];
    else
      A[k++] = C[j++];
  }
  for ( ; i<n; i++)   A[k++] = B[i];  //남은 B[]의 데이터 → A[]
  for ( ; j<m; j++)  A[k++] = C[j];  //남은 C[]의 데이터 → A[]
  return A[0..n+m-1];
}
```

합병 함수 Merge()의 동작

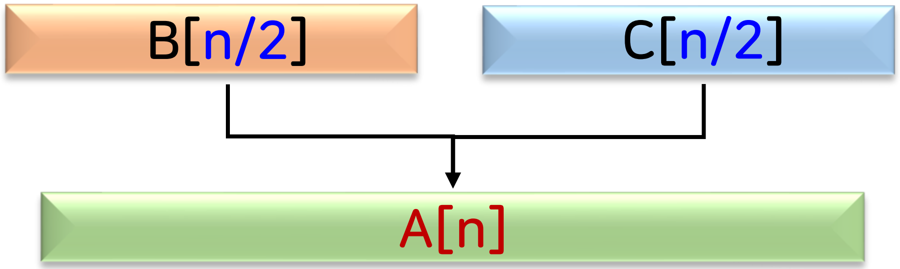


합병 함수 Merge()의 동작

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

성능 분석

합병 함수 Merge() 수행 시간



두 부분배열 간의 비교 횟수 $\frac{n}{2} \sim (\frac{n}{2}+\frac{n}{2}-1=n-1)$

최악의 경우

Θ(n)

- 입력 데이터 개수 n만큼의 추가 저장 장소(B[]+C[])가 필요

성능 분석

합병 정렬 MergeSort() 수행 시간

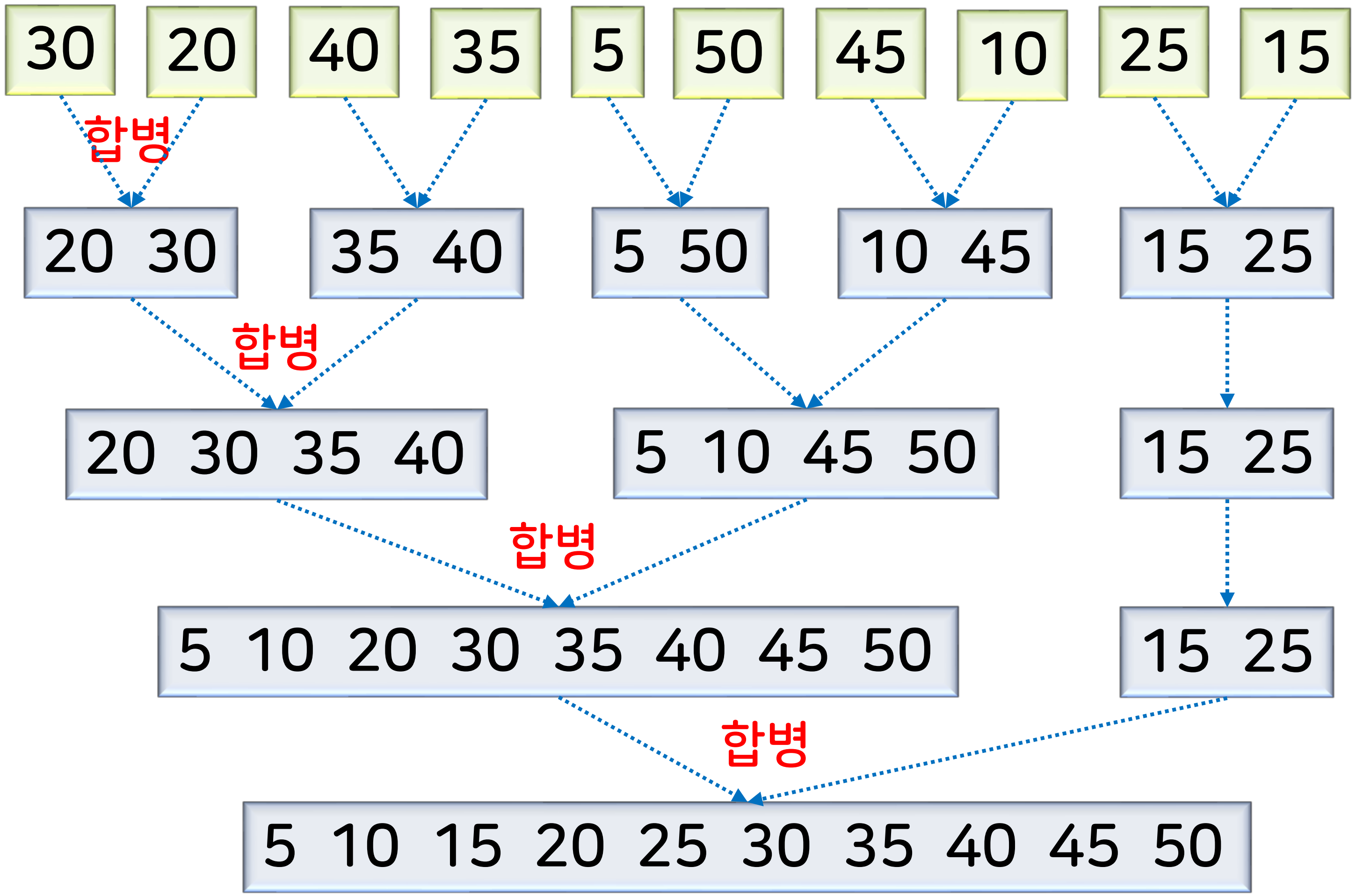
```
MergeSort (A[ ], n)                                          → T(n)
{
  if (n > 1) {
    Mid = ⌊n/2⌋;
    B[0..Mid-1] = MergeSort(A[0..Mid-1], Mid);               → T(⌊n/2⌋)
    C[0..n-Mid-1]  = MergeSort(A[Mid..n-1], n-Mid);          → T(⌊n/2⌋)
    A[0..n-1] = Merge(B[0..Mid-1], C[0..n-Mid-1], Mid,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

$T(n) = T(\lfloor n/2 \rfloor) + T(\lfloor n/2 \rfloor) + \Theta(n)$ $(n>1)$, $T(1)=1$

$T(n) = 2T(n/2) + \Theta(n)$

$T(n) = O(n\log n)$

비순환적 합병 정렬



30 20 40 35 5 50 45 10 25 15
합병
20 30
35 40
5 50
10 45
15 25
합병
20 30 35 40
5 10 45 50
15 25
합병
5 10 20 30 35 40 45 50
15 25
합병
5 10 15 20 25 30 35 40 45 50

02

선택 문제

개념과 원리

선택 문제?

- n개의 원소가 임의의 순서로 저장된 배열 A[0..n-1]에서 i 번째로 작은 원소를 찾는 문제
  - ✓ i=1 → 최솟값, … , i=$\lfloor n/2 \rfloor$ → 중간값, … , i=n → 최댓값
- 직관적인 방법
  - ✓ 오름차순으로 정렬한 후 i번째 원소를 찾는 방법 → O(nlogn)
  - ✓ 최솟값을 찾는 과정을 i번 반복(i-1번째까지는 최솟값을 찾은 후 삭제) → O(in)
- 알고리즘① → 최악 $O(n^2)$, 평균 O(n) 알고리즘
- 알고리즘② → 최악 O(n), 평균 O(n) 알고리즘

최솟값 찾기

각 데이터를 하나씩 모두 비교하는 방법

- n개의 데이터에 대해서 적어도 (n-1)번의 비교가 필요 → O(n)

```
FindMinimum (A[], n)
{
   min = A[0];
   for (i=1; i<n; i++)
      if (A[i]<min) min = A[i];
   return min;
}
```

최솟값과 최댓값 모두 찾기

- **방법1:** 최솟값 찾은 후 최댓값 찾기 (or 최댓값 찾은 후 최솟값 찾기)
  - n개의 데이터에서 최솟값을 찾는데 (n-1)번 비교

    \+ (n-1)개의 데이터에서 최댓값을 찾는데 (n-2)번 비교

    ⇒ 2n-3번의 비교

- **방법2:** 모든 원소를 두 개씩 짝을 이루어
  동시에 최솟값/최댓값과 비교

  ⇒ $\frac{3}{2}n-2$번의 비교

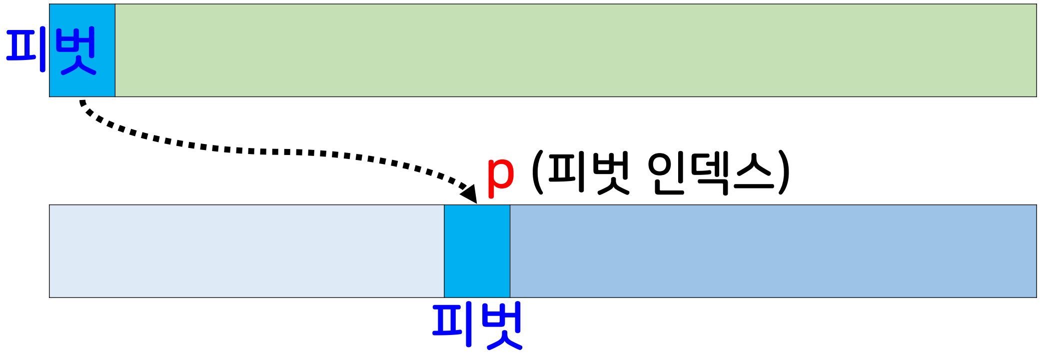
최솟값과 최댓값 모두 찾기

```
FindMinMax (A[], n, min, max)
{
   if (A[0]<A[1]) {  min = A[0];  max = A[1];  }
   else {  min = A[1];  max = A[0];  }
   for (i=2; i<n; i+=2) {
      if (A[i]<A[i+1]) {  small = A[i];   large = A[i+1];  }
      else {  small = A[i+1];   large = A[i];  }
      if (small < min) min = small;
      if (large > max) max = large;
   }
}
```

i번째로 작은 원소 찾기_최악 $O(n^2)$, 평균 $O(n)$

개념과 원리

- 퀵 정렬의 분할 함수 Partition()을 순환적으로 적용하는 방법



✓ $i = p$ → 피벗이 찾고자 하는 i번째 원소

✓ $i < p$ → 왼쪽 부분배열에 대해 순환 적용

✓ $i > p$ → 오른쪽 부분배열에 대해 순환 적용

i번째로 작은 원소 찾기_최악 $O(n^2)$, 평균 $O(n)$

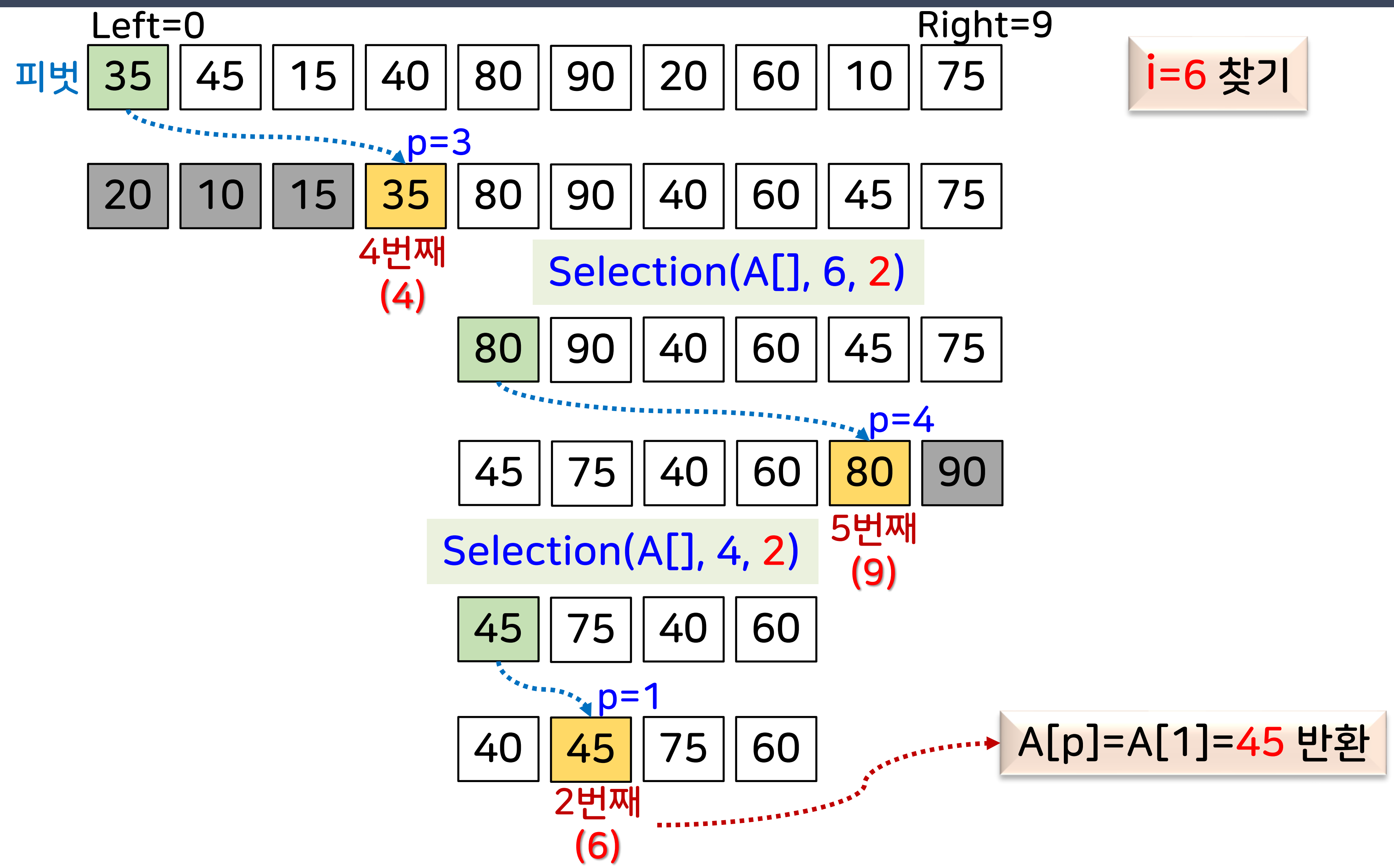
개념과 원리

분할	피벗을 기준으로 주어진 배열을 두 부분배열로 분할하고, i가 피벗의 인덱스 p와 같으면 피벗의 값을 반환하고 종료
정복	인덱스 i가 포함된 부분배열에 대해서 알고리즘을 순환적으로 호출하여 적용
결합	필요 없음

i번째로 작은 원소 찾기_최악 $O(n^2)$, 평균 $O(n)$

```
int Selection (A[], n, i)   // 1 ≤ i ≤ n
{
   Left = 0;  Right = n-1;
   p = Partition(A, n);     // 0 ≤ p ≤ n-1
   if (i == p+1) return A[p];
   else if ( i<p+1 ) Selection(A[Left..p-1], (p-1)-Left+1, i);
         else Selection(A[p+1..Right], Right-(p+1)+1, i-p-1);
}
```

i번째로 작은 원소 찾기_최악 $O(n^2)$, 평균 $O(n)$



i번째로 작은 원소 찾기_최악 $O(n^2)$, 평균 $O(n)$

성능 분석

- 최악의 경우 = 퀵 정렬의 최악의 경우
  - ✓ 분할 함수 Partition()이 항상 하나의 부분배열만 생성하는 경우
  - ✓ 오름차순을 정렬된 상태에서 최댓값(i=n)을 찾는 경우
    - → 분할 함수를 호출할 때마다 피벗 인덱스는 1씩 증가
    - → Partition()를 O(n)번 호출 ⇒ $O(n^2)$
  - ✓ 해결 방법 → 항상 일정한 비율의 두 부분배열로 분할 → 최악의 경우에도 O(n)
- 평균적인 경우 → O(n)

i번째로 작은 원소 찾기_최악 O(n), 평균 O(n)

개념과 원리

- 항상 일정한 비율의 두 부분배열로 분할되도록
  특정 성질을 만족하는 값을 피벗으로 선택

피벗 선택 방법

(1) 크기 n인 배열의 원소를 5개씩 묶어서 $\lfloor n/5 \rfloor$개의 그룹을 만듦

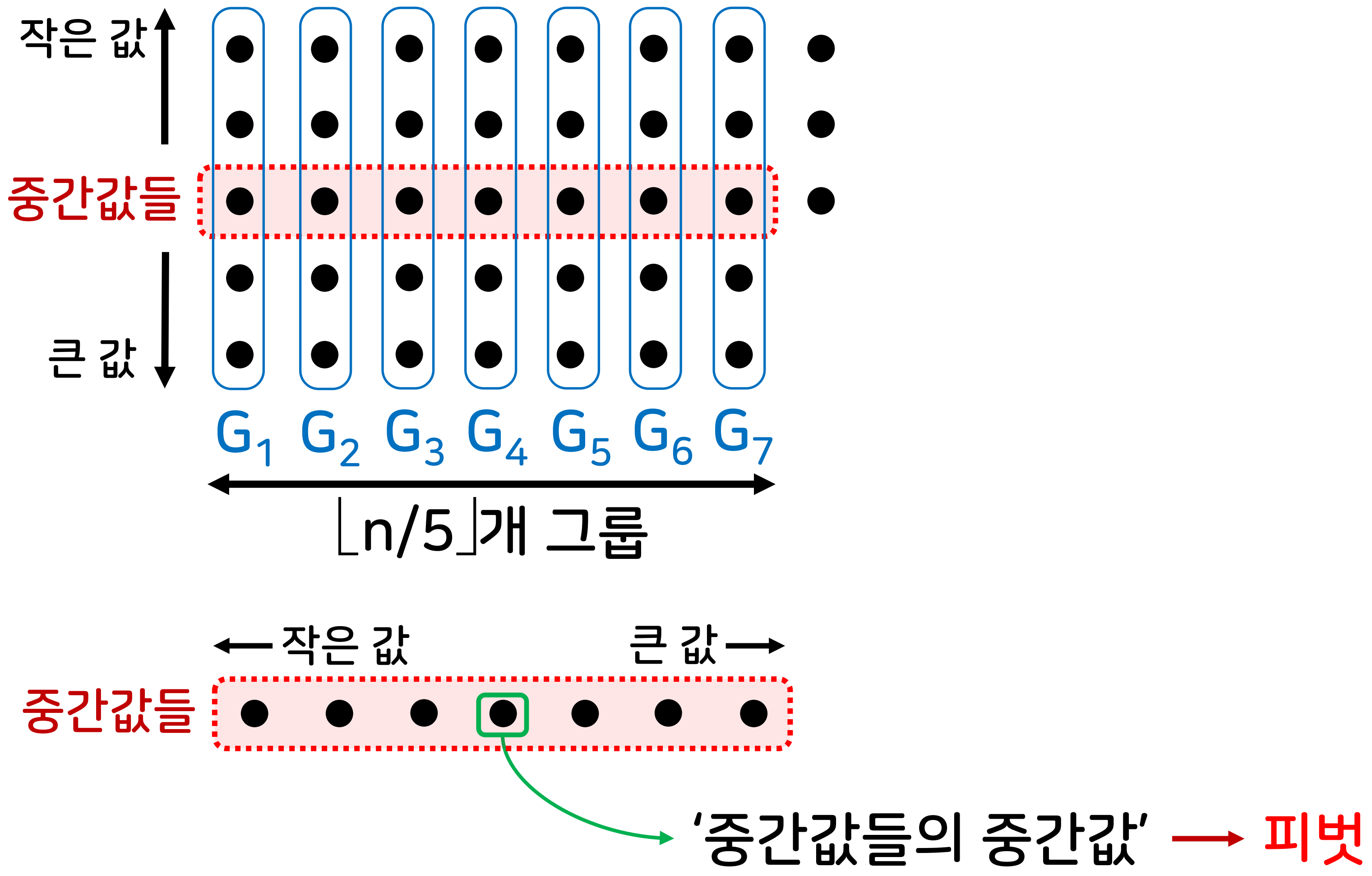
✓ 5의 배수가 되지 않아 그룹을 형성하지 못한 채 남는 원소는 그대로 남겨 둠

(2) 각 그룹에 대해서 중간값을 찾음

(3) $\lfloor n/5 \rfloor$개의 중간값을 대상으로 다시 중간값을 찾음

→ "중간값들의 중간값" → 피벗

i번째로 작은 원소 찾기_최악 O(n), 평균 O(n)



i번째로 작은 원소 찾기_최악 O(n), 평균 O(n)

A[]={ 9, 6, 35, 39, 15,   24, 70, 95, 50, 1,   97, 84, 77, 28, 10,   22, 27, 11, 31, 62,
54, 81, 5, 34, 4,   89, 60, 29, 2, 75,   18, 36, 80, 7, 53,   25, 66, 43 }

9	24	97	22	54	89	18	25
6	70	84	27	81	60	36	66
35	95	77	11	5	29	80	43
39	50	28	31	34	2	7	
15	1	10	62	4	75	53	
G_1	G_2	G_3	G_4	G_5	G_6	G_7	

그룹별 정렬

6	1	10	11	4	2	7	25
9	24	28	22	5	29	18	66
15	50	77	27	34	60	36	43
35	70	84	31	54	75	53	
39	95	97	62	81	89	80	
G_1	G_2	G_3	G_4	G_5	G_6	G_7	

중간값들

정렬

15	27	34	36	50	60	77
G_1	G_4	G_5	G_7	G_2	G_6	G_3

중간값들의 중간값 → 피벗

i번째로 작은 원소 찾기_최악 O(n), 평균 O(n)

부분배열로의 분할 비율

S_1

G_1	G_4	G_5	G_7	G_2	G_6	G_3	
6	11	4	7	1	2	10	25
9	22	5	18	24	29	28	66
15	27	34	36	50	60	77	43
35	31	54	53	70	75	84	
39	62	81	80	95	89	97	

S_2

- S_1과 S_2에 각각 속하는 원소의 개수 = $3 \times \lceil \lfloor n/5 \rfloor / 2 \rceil$

  → 분할할 때마다 탐색 대상에서 제외되는 데이터 개수

i번째로 작은 원소 찾기_최악 O(n), 평균 O(n)

```
int Selection_n (A[], n, i)  {
  [단계1]  if ( n <= 5 ) 배열 A에서 i번째 원소를 찾아서 반환
           else [단계2]~[단계6]을 진행
  [단계2]  A의 원소를 5개씩 묶어서 ⌊n/5⌋개의 그룹을 생성
  [단계3]  각 그룹의 중간값을 구하고, 이들을 모아 배열 M을 구성
  [단계4]  중간값들의 중간값을 계산하기 위해서 선택 함수를 순환 호출
            p = Selection_n ( M, ⌊n/5⌋, ⌈⌊n/5⌋/2⌉)
  [단계5]  p를 피벗으로 사용하여 A를 분할 (피벗의 인덱스 = j라고 가정)
  [단계6]  if ( i == j+1 ) return A[j]
           else if ( i < j+1 ) Selection_n(A[0..j-1], j, i)를 순환 호출
                else Selection_n(A[j+1..n-1], n-j-1,i-j-1)를 순환 호출
}
```

오늘의 학습

정 리 하 기

정리하기

1. 합병 정렬

1) 같은 크기의 두 부분배열로 분할하고 순환 호출하여 정렬
   + 정렬된 두 부분배열을 합쳐서 하나의 정렬 배열을 만듦
   ① 합병 함수 Merge() → 정렬된 두 부분배열을 하나의 정렬 배열로 만드는 함수
   ② 성능: $T(n)=2T(n/2)+\Theta(n)$, $T(1)=\Theta(1)$ → $O(n\log n)$
   ③ 입력 크기 n만큼의 추가적인 저장 장소 필요

2. 선택 문제

1) 임의의 순서로 주어진 n개의 원소에서 i번째로 작은 원소를 찾는 문제
2) 최솟값/최댓값 찾기
   ① 최소값 찾기 → 적어도 (n-1)번의 비교 → $O(n)$
   ② 최소값과 최대값 모두 찾기 → 3n/2 -2 번 비교 → $O(n)$
3) 선택 문제 알고리즘
   ① 퀵 정렬의 Partition() 이용 → 최악 $O(n^2)$, 평균 $O(n)$
   ② "중간값들의 중간값" 이용 → 최악 $O(n)$, 평균 $O(n)$

다음시간에는

5강

동적 프로그래밍 알고리즘 (1)