알고리즘 시작하기

소프트웨어학부
박영훈 교수

정렬 문제

- 알고리즘의 여러 가지 문제들 중 가장 기본적이면서도 많이 사용되는 것이 정렬 문제이다.
- 정렬 문제는 다음과 같이 정의한다:
    - 입력: n개 수로 이루어진 수열 $\{a_1, a_2, \cdots, a_n\}$
    - 출력: $a'_1 \le a'_2 \le \cdots \le a'_n$, $\{a_1, a_2, \cdots, a_n\} = \{a'_1, a'_2, \cdots, a'_n\}$을 만족하는 입력 수열의 재배열 a'_1, a'_2, $\cdots$, a'_n.
- 본 chapter에서는 정렬 알고리즘 중 가장 기본적인 것인 insertion sort와 merge sort에 대하여 알아볼 것이다.
- 또한, 각 알고리즘의 수행 시간도 분석해볼 것이다.

Insertion Sort (삽입정렬), 사람이 본능적으로 sorting 하는 방식.

- 두 번째 원소부터 맨 마지막 원소까지에 대하여, 각 원소가 왼쪽에 있는 모든 원소들보다 크거나 같을 때까지 왼쪽으로 이동시킴으로써 정렬한다.

- 예: 5, 2, 4, 6, 1, 3이 insertion sort로 정렬되는 과정은 다음과 같다.

두 번째 원소의 이동	5	**2**	4	6	1	3	→	**2**	5	4	6	1	3
세 번째 원소의 이동	2	5	**4**	6	1	3	→	2	**4**	5	6	1	3
네 번째 원소의 이동	2	4	5	**6**	1	3	→	2	4	5	**6**	1	3
다섯 번째 원소의 이동	2	4	5	6	**1**	3	→	**1**	2	4	5	6	3
여섯 번째 원소의 이동	1	2	4	5	6	**3**	→	1	2	**3**	4	5	6

Insertion Sort Algorithm

- 앞의 내용을 바탕으로 insertion sort 과정을 pseudo code로 나타내면 다음과 같다:

```
Function insertionSort(A, n){
    for j = 1 to n − 1{   수열의 처음부터 마지막까지 반복
        key ← A[j];
        i ← j − 1;
        while(i ≥ 0 && A[i] > key){   맨 왼쪽에 오거나, 작은게 등장하면 stop
            A[i + 1] ← A[i];
            i ← i − 1;   한칸씩 앞으로 이동
        }
        A[i + 1] ← key;
    }
}
```

알고리즘 분석을 위한 가정

- 모든 명령어는 위에서부터 순차적으로 실행된다. Multi-threading 과 같은 효과는 고려하지 않는다.

  동시수행

- 모든 메모리계층은 입출력 속도가 같다고 가정한다.

- 실제 상황에서는 각 명령어의 비용 및 수행 시간이 다르다. (예를 들어, 덧셈과 나눗셈의 수행 시간은 다르며, 변수에 값을 저장하는 것(예: $a \leftarrow x$)과 배열 변수에 값을 저장하는 것(예: $a[i] \leftarrow x$)은 소요되는 시간 역시 다르다) 하지만, 이런 요소들을 모두 고려하게 되면 분석이 너무 복잡해지며, 수행 시간 분석에 중요한 요소는 아니므로 무시한다.

  메모리 이동 시간이 추가되지만 고려X

- 사용되는 수의 크기에 상관 없이 계산 시간은 모두 같다고 가정한다.

Insertion Sort의 수행 시간 분석

- Insertion sort 수행 시 소요 시간을 측정하기 위하여, 다음과 같이 Cost를 정하고, 반복 횟수를 구해본다.

*각 라인이 몇 번 수행되는가

		Cost	반복횟수
1:	**Function** *insertionSort*$(A, n)\{$		
2:	**for** $j = 1$ **to** $n - 1\{$ 조건 검사 1되까지 n번수행	c_1	n
3:	$key \leftarrow A[j];$	c_2	$n-1$
4:	$i \leftarrow j - 1;$ $n-1$	c_3	$n-1$
5:	**while**$(\boxed{i \geq 0}\ \&\&\ \boxed{A[i] > key})\{$ t_j 중간 break	c_4	t_j 5번라인 실행횟수
6:	제일 많이 수행되는 경우 $A[i+1] \leftarrow A[i];$ $t_j - 1$	c_5	$t_j - 1$
7:	$j-1$ to -1 $i \leftarrow i - 1;$	c_6	$t_j - 1$
8:	$\underbrace{= j+1}_{t_j \text{ 상한}}$ $\}$		
9:	$A[i+1] \leftarrow key;$ $n-1$	c_7	$n-1$
10:	$\}$		
11:	$\}$		

Insertion Sort의 수행 시간 분석

- 위의 pseudo-code에서 반복문의 반복 조건(line 2, line 5)을 수행하는 횟수는 반복 내용을 수행하는 횟수보다 1회 많다. 그 이유는 반복문을 빠져나갈 때 반복 조건을 한 번 더 수행하기 때문이다.

- 각각의 j에 대하여, line 5가 수행되는 횟수를 t_j라 하였다. 그러면 실제로 line 5가 실행되는 횟수는 $\sum_{j=1}^{n-1} t_j$가 된다.

- Insertion sort를 수행하는데 걸리는 시간을 $T(n)$이라 하자. 그러면 $T(n)$은 다음과 같이 계산된다:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + \sum_{j=1}^{n-1} \left\{ c_4 t_j + c_5(t_j - 1) + c_6(t_j - 1) \right\} + c_7(n-1)$$

$$= (c_4 + c_5 + c_6) \sum_{j=1}^{n-1} t_j + (c_1 + c_2 + c_3 - c_5 - c_6 + c_7)n + (-c_2 - c_3 + c_5 + c_6 - c_7)$$

Insertion Sort의 수행 시간 분석 (Cont'd)

- 한편, line 5-8 반복문에 대하여, 초기 i값은 $j-1$이고, 반복되는 동안 0 이상을 유지하므로, 최대 $j+1$회 반복된다. 따라서 $t_j \leq j+1$이라 할 수 있다. $\therefore \sum_{j=1}^{n-1} t_j \leq \frac{n(n+1)}{2} - 1$.

n에 대한 이차식으로 정리

$$\therefore T(n) \leq \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$$

- 즉, $T(n)$의 최댓값은 $an^2 + bn + c$의 형태로 나타낼 수 있다.

Insertion Sort의 최선의 경우와 최악의 경우

- 수열의 길이가 고정되어 있을 때, 수행 시간에 영향을 줄 수 있는 요소는 수열의 최초 배열 상태이다.
- 수열이 최초에 어떻게 배열되어있는지에 따라서 t_j의 값이 달라진다.
- 최선의 경우 오름차순 정렬된경우 n에대한 일차식이된다
  - 모든 j에 대하여, t_j의 값이 1이 될 수 있다면 이 경우가 바로 최선이 된다.
  - 이미 오름차순으로 배열되어 있다면, line 5의 $A[i] > key$ 조건이 거짓이 되므로 line 5-8의 반복문이 실행되지 않는다. 따라서 이 경우가 $t_j = 1$이 되는 경우이다.
  - 이 경우, $T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$이 된다.
- 최악의 경우 역순 정렬된 경우 n에 대한 이차식
  - 모든 j에 대하여, t_j의 값이 $j + 1$이 될 수 있다면 이 경우가 바로 최악이 된다.
  - 배열이 내림차순으로 배열되어 있다면, line 5의 $A[i] > key$ 조건이 항상 참이 되며, i가 -1이 될 때까지 반복된다. 즉, 반복 횟수가 $j + 1$회가 되는데, 이 경우가 최악의 경우가 된다.
  - 이 때,

$$T(n) = \frac{c_4 + c_5 + c_6}{2} n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7 \right) n - (c_2 + c_3 + c_4 + c_7)$$

이다.

※ 차수만 중요해짐.

증가 차수

- Insertion sort의 최악의 경우에서 수행 시간은 배열의 크기가 n일 때, $an^2 + bn + c$의 형태로 나타낼 수 있었다.
- 위 수식에서 가장 중요한 항은 최고차항, 즉 an^2이다. 또한, 최고차항의 차수가 더욱 중요하므로 n^2만 남긴다.
- 이 경우, 우리는 $\Theta(n^2)$의 수행시간이 걸린다고 말한다.

  세타(n^2) 수행시간이 걸린다

- 마찬가지로, insertion sort의 최선의 경우의 수행 시간은 $dn + e$ 형태로 나타낼 수 있다.
- 이 경우, 우리는 $\Theta(n)$의 수행시간이 걸린다고 할 수 있다.
- Θ의 정확한 정의는 다음에 다룰 것이다.
- 수행 시간이 오래 걸리는 순서는 다음과 같다:

대부분 정렬 / 최댓값찾기 / 소인수분해

$$\cdots > \Theta(n^3) > \Theta(n^2) > \Theta(n^{1.xxx}) > \Theta(n \log n) > \Theta(n) > \Theta(\sqrt{n}) > \Theta(\log n)$$
$$> \Theta(\log \log n) > \Theta(1)$$

Θ 함수의 특징

- 어떤 알고리즘의 수행 시간이 $a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$일 때, 수행 시간을 최고 차항을 제외한 나머지 항들과 계수들을 빼버리고 $\Theta(n^k)$로만 표기한다.
  - $\frac{\log n}{\log 2}$ 만일 수행 시간이 $b \log_2 n + a$이면, $\Theta(\log n)$ 으로 표기.
  - 만일 수행 시간이 $a_1 n + b \log n + a_0$ 이면, $\Theta(n)$ 으로 표기.
  - 만일 수행 시간이 $bn \log n + a_1 n + a_0$ 이면, $\Theta(n \log n)$ 으로 표기.

- 위의 성질들을 이용하여 다음이 성립함을 알 수 있다.
  - 어떤 n에 대한 함수 $f(n)$이 있을 때, 임의의 양수 c에 대하여, $c \times \Theta(f(n)) = \Theta(f(n))$.
  - 어떤 n에 대한 두 함수 $f(n)$, $g(n)$에 대하여, $f(n)$의 차수가 $g(n)$의 차수보다 클 때,

    $\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n))$, $\Theta(f(n)) + g(n) = \Theta(f(n))$, $f(n) + \Theta(g(n)) = \Theta(f(n))$
  - 어떤 n에 대한 두 함수 $f(n)$, $g(n)$에 대하여, $\Theta(f(n)) \times g(n) = \Theta(f(n)g(n))$.

수행 시간 간편 분석 방법

- Pseudo code가 주어진 상태에서 수행 시간을 분석하기 위해서는 각 line이 몇 번씩 실행되는지를 분석하면 된다.

- 하지만, 이렇게 하면 정확하게 구할 수 있지만, 시간이 오래 걸릴 뿐 아니라 실제로 표현할 때는 수행 시간의 최고 차항이 몇 차인지만 중요하기 때문에 쓸데 없는 에너지 낭비가 될 것이다.

- 한편, 수행 시간 식의 각각의 항들은 반복문들에 의해 결정된다. 또한, 최고 차항은 가장 많은 겹의 반복문에 의해 결정된다.

- 따라서 수행 시간을 분석할 때는 가장 많은 겹의 반복문만 관찰하면 된다.

수행 시간 간편 분석 방법 (Cont'd)

1: **Function** $insertionSort(A, n)\{$

2: $\quad$ **for** $j = 1$ **to** $n - 1\{$

3: $\quad\quad key \leftarrow A[j];$

4: $\quad\quad i \leftarrow j - 1;$

$\Theta(n)$번 반복된다.

5: $\quad\quad$ **while**$(i \geq 0 \;\&\&\; A[i] > key)\{$

6: $\quad\quad\quad A[i+1] \leftarrow A[i];$

7: $\quad\quad\quad i \leftarrow i - 1;$

8: $\quad\quad \}$

이 부분에 의해서 수행 시간이 결정된다.

9: $\quad\quad A[i+1] \leftarrow key;$ $\longrightarrow$ $\Theta(n)$번 반복된다.

10: $\quad \}$

11: $\}$

- Line 5-8이 몇 번씩 반복되는지 세면 되는데, line 5가 line 6, 7, 8보다 1번씩 더 많이 실행되므로 line 5만 몇 번 반복되는지 보면 된다.
- 최선의 경우는 반복할 때마다 Line 5가 1번씩 실행되고, 최악의 경우는 Line 5가 j번 반복된다.
- 평균적인 경우는...?

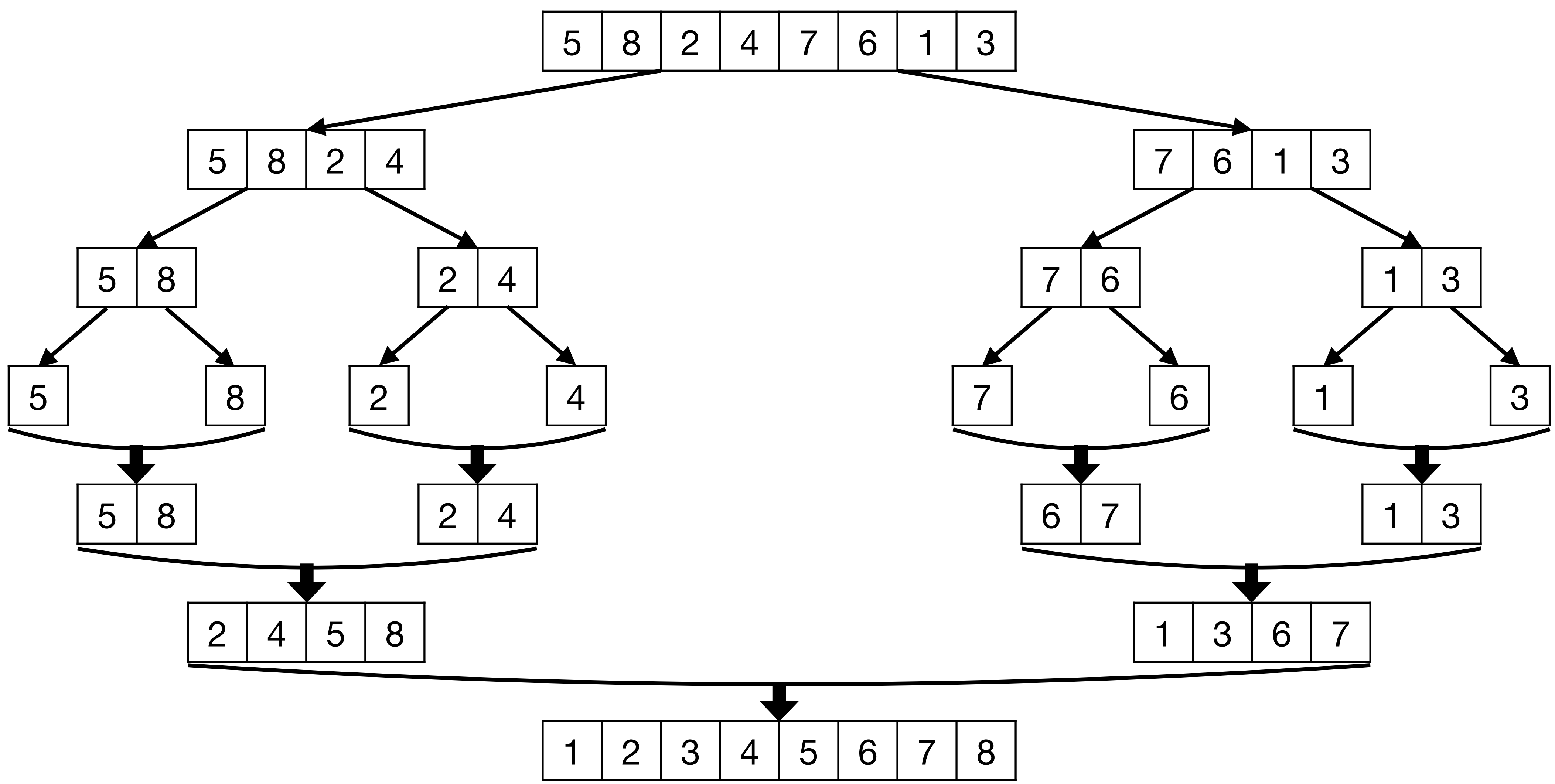
Divide and Conquer

- 원래 문제를 작은 문제로 분할하여, 더이상 분할할 수 없을 때 까지 나눈 뒤, 그 분할된 요소들을 처리하고, 처리된 결과를 다시 합침으로써 해결하는 방법.
- 평균적인 경우, 수행 시간이 $\Theta(n^2)$이 걸리는 문제를 $\Theta(n \log n)$으로 줄여주기 때문에 정렬 문제에 매우 널리 사용된다.
- Divide and conquer 를 프로그램으로 구현하기 위하여 재귀함수를 주로 사용한다.
  - 나누어진 작은 문제에 대하여 같은 루틴을 적용해야 하기 때문.
- Divide and conquer는 주로 다음과 같은 단계들로 구성된다.
  - Divide: 현재 문제를 같은 문제를 다루는 다수의 부분 문제로 분할하는 과정
  - Conquer: 더이상 분할될 수 없는 작은 문제들을 해결하는 과정.
  - Combine: 해결된 작은 문제들을 다시 합쳐서 원래 문제의 답이 되도록 만드는 과정.

Merge Sort

- Divide and conquer 를 기반으로 하는 가장 기본적인 정렬 기법.
  - Divide: 정렬할 배열을 절반씩 나눈다.
  - Conquer: 하는 일 없음
  - Combine: 절반씩 나뉜 부분 배열들을 다시 합침과 동시에 정렬을 수행한다.

- 실행 예:



Merge Sort Algorithm

- Merge sort를 수행하기 위한 알고리즘은 재귀 함수 기반으로 만들어진다. 수열 $A[]$의 p번째 항부터 r번째 항까지 오름차순 배열하기 위하여, 다음과 같이 만들 수 있다:

배열

Function $mergeSort(A, \underline{p, r})$ { p to r 배열 (값)

  if$(p < r)${ 원소가2개이상

    $q \leftarrow \lfloor(\underline{p + r})/2\rfloor$; 올림.

    평균

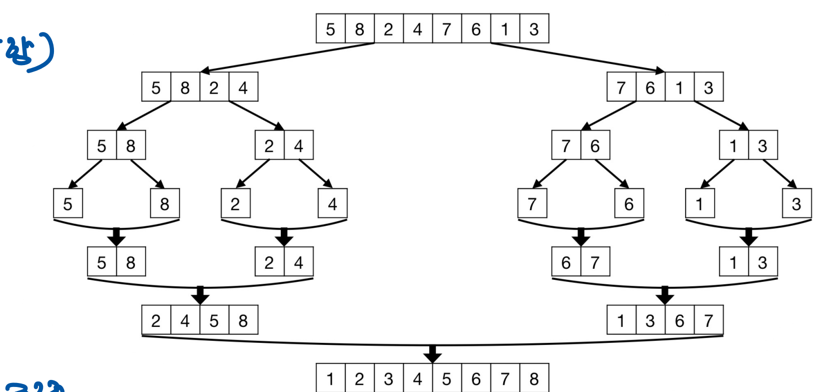
    $mergeSort(A, p, q)$; 왼쪽

    $mergeSort(A, q + 1, r)$; 오른쪽

    $merge(A, p, q, r)$; 합하기

  }

}



- 길이가 n인 수열 $A[]$를 정렬하기 위하여 위의 함수를 최초로 부를 때는 다음과 같이 하면 된다:

$$\underline{mergeSort(A, 0, n - 1);}$$

Merge 함수

- Merge 함수의 pseudo code는 다음과 같다:

$\frac{n}{2}$

0 $n-1$ $\qquad$ $\frac{n}{2}+1$

Function $merge(A, p, q, r)\{$

$\quad n_1 \leftarrow q - p + 1, n_2 \leftarrow r - q;$ ← $\Theta(1)$

$\quad$ Create two arrays $L[0 \cdots n_1]$ and $R[0 \cdots n_2]$; ← $\Theta(n)$

$\quad$ **for** $i = 0$ **to** $n_1 - 1$ $\{L[i] \leftarrow A[p+i];\}$ ← $\Theta(n)$

$\quad$ **for** $j = 0$ **to** $n_2 - 1$ $\{R[j] \leftarrow A[q+j+1];\}$ ← $\Theta(n)$

$\quad L[n_1] \leftarrow \text{inf}, R[n_2] \leftarrow \text{inf};$ ← $\Theta(1)$

$\quad i \leftarrow 0, j \leftarrow 0;$

$\quad$ o to

$\quad$ **while**$(i < n_1 \,\|\, j < n_2)\{$ $\qquad$ $i+j$는 1증가 0 ~ n ← $\Theta(n+1)$

$\qquad$ if$(L[i] < R[j])$ $\quad \{A[p+i+j] \leftarrow L[i];$ $\quad i{+}{+};\}$

$\qquad$ else $\quad \{A[p+i+j] \leftarrow R[j];$ $\quad j{+}{+};\}$

$\quad \}$

$\}$

$\Theta(n)$

Merge 함수의 동작 과정

정수: $2^{31}-1 = 111\cdots1_{(2)}$ = 7fffffff

- Merge 함수는 Combine 과정으로, merge 함수에 입력으로 들어가는 두 부분 수열은 이미 정렬이 되어 있다.
- 수열 $A[] = \{2,4,5,8,1,3,6,7\}$에서, $merge(A,0,3,7)$; 를 실행하면 다음과 같은 과정으로 정렬된다.

$A[0\ \cdots\ 7]$

2	4	5	8	1	3	6	7
1	4	5	8	1	3	6	7
1	2	5	8	1	3	6	7
1	2	3	8	1	3	6	7
1	2	3	4	1	3	6	7
1	2	3	4	5	3	6	7
1	2	3	4	5	6	6	7
1	2	3	4	5	6	7	7
1	2	3	4	5	6	7	8

i+j 번째

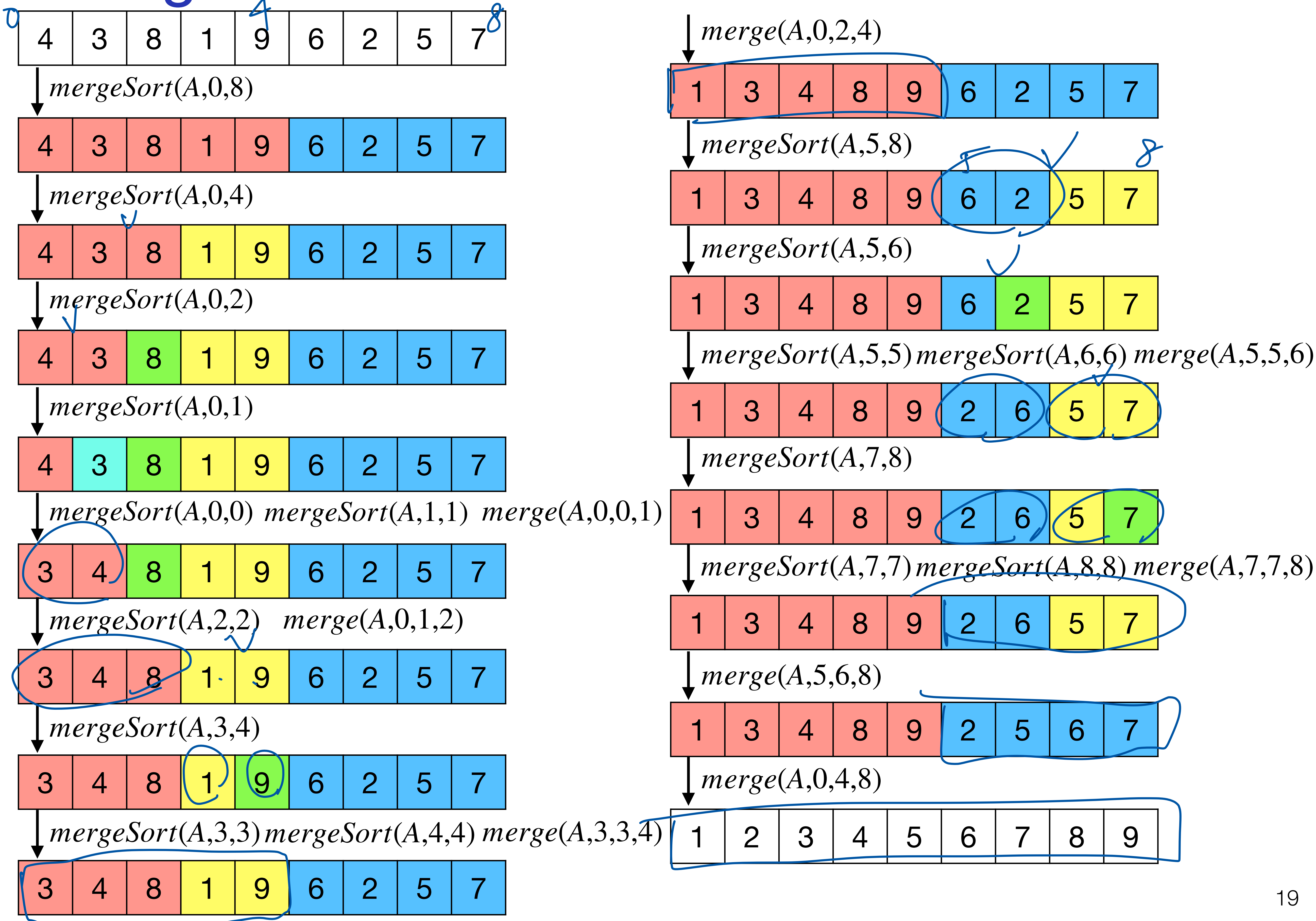
$L[0\ \cdots\ 4]$

2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞
2	4	5	8	∞

$R[0\ \cdots\ 4]$

1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞
1	3	6	7	∞

Merge Sort 동작 예



Merge Sort의 수행 시간 분석

- 길이가 n인 배열을 merge sort하는 데 걸리는 시간을 $T(n)$이라고 하자.
- 또한, $k = r - p + 1$일 때(즉, k는 합쳐질 수열의 길이), $merge$ 함수를 실행하는데 걸리는 시간을 $C(k)$라 하자.
- 그러면 merge sort algorithm에 의해, $T(n)$을 다음과 같이 계산할 수 있을 것이다:

$$T(n) = 2T(n/2) + C(n).$$

- 한편, $merge$함수에서 line 4와 5는 각각 $\Theta(n_1)$, $\Theta(n_2)$의 수행 시간이 걸리고, line 8, 9, 10은 각각 $\Theta(r-p)$의 수행시간이 걸리며, 나머지 line들은 각각 $\Theta(1)$의 수행시간이 걸린다.
- $n_1, n_2 < r - p = k - 1$이므로, $C(k) = \Theta(k)$. $C(n) = \Theta(n)$
- 따라서, $T(n)$을 구하기 위한 점화식은 다음과 같이 정리된다:

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

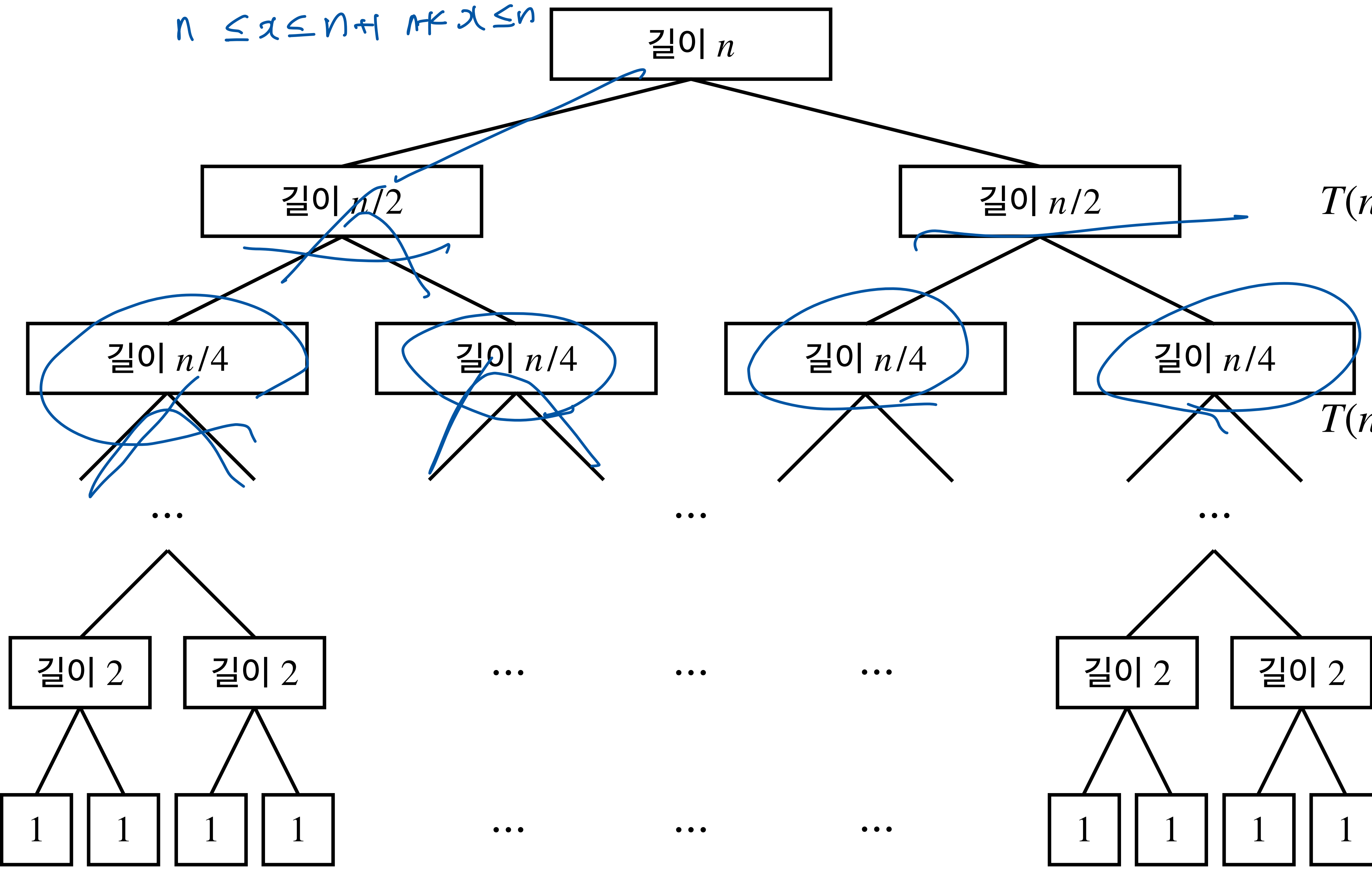
Merge Sort의 수행 시간 분석 (Cont'd)

$h-1 < \log_2 n \le h$

- $\boxed{h = \lceil \log_2 n \rceil}$이라고 하자. 그러면, $\boxed{2^{h-1} < n \le 2^h}$이 되며, leaf node가 n개인 binary tree를 만들면 높이가 h가 된다.

$\lfloor x \rfloor = n$ $\qquad \lceil x \rceil = n$

$n \le x \le n+1 \qquad n-1 < x \le n$



수행 시간

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

$T(n/2) = 2T(n/4) + \Theta(n/2)$

$T(n/4) = 2T(n/8) + \Theta(n/4)$

…

$T(2) = 2T(1) + \Theta(2)$

$\boxed{T(1) = \Theta(1)}$

Merge Sort의 수행 시간 분석 (Cont'd)

$T(n/2) = 2T(n/4) + \Theta(\frac{n}{2})$

$$
\begin{aligned}
\therefore T(n) &= 2T(n/2) + \Theta(n) \\
&= 4T(n/4) + 2\Theta(n/2) + \Theta(n) \\
&= 8T(n/8) + 4\Theta(n/4) + 2\Theta(n/2) + \Theta(n) \\
&\qquad \vdots \\
&= 2^h T(\underbrace{n/2^h}_{\leq 1}) + 2^{h-1}\Theta(n/2^{h-1}) + 2^{h-2}\Theta(n/2^{h-2}) + \cdots + 2\Theta(n/2) + \Theta(n) \\
&\leq 2^h \Theta(n/2^h) + 2^{h-1}\Theta(n/2^{h-1}) + 2^{h-2}\Theta(n/2^{h-2}) + \cdots + 2\Theta(n/2) + \Theta(n) \\
&= \underbrace{\Theta(n) + \Theta(n) + \cdots + \Theta(n)}_{h+1 \text{ terms}} \\
&= (h+1)\Theta(n) \\
&= \log_2 n \times \Theta(n) + \Theta(n) \\
&= \Theta(n \log n)
\end{aligned}
$$

어느정도 정렬되었거나 수가 적으면 incertion sort가 좋음

- 따라서 길이가 n인 수열을 merge sort를 이용하여 오름차순으로 배열하는데 걸리는 수행 시간은 $\Theta(n \log n)$이 된다.