

*2주차 : Insertion & Merge sort*

# 알 고 리 즈

2017. 9. 14.

충남대학교 컴퓨터공학과 분산이동컴퓨팅 연구실  
조교 이정진

# Overview

- ▶ 알고리즘의 수행 시간

- 1) 시간 복잡도

- 2)  $O$ -,  $\Omega$ -, and  $\Theta$ -notation

- ▶ 두 가지 정렬 방법 소개 및 Time Complexity 계산

- 1) Insertion sort

- 2) Merge sort

- ▶ 실습 / 과제

- 파일 입출력을 사용한 Insertion & Merge sort 구현

# Time Complexity

## ▶ Time Complexity (시간 복잡도)

알고리즘을 구성하는 모든 명령어들에 대해서  
각각의 [ 수행에 필요한 Cost x 수행 횟수 ] 의 총합



# Notation

▶ **O-notation (최악의 경우) :  $f(n) = O(g(n))$**

모든  $n \geq n_0$ 에 대해  $0 \leq f(n) \leq cg(n)$ 인 양의 상수  $n, c$ 이 존재할 때  
e.g.  $2n^2 = O(n^2)$  ( $c=1, n_0=2$ )

▶  **$\Omega$ -notation (최상의 경우) :  $f(n) = \Omega(g(n))$**

모든  $n \geq n_0$ 에 대해  $0 \leq cg(n) \leq f(n)$ 인 양의 상수  $n, c$ 이 존재할 때  
e.g.  $\sqrt{n} = \Omega(\lg n)$  ( $c=1, n_0=16$ )

▶  **$\Theta$ -notation (평균인 경우) :  $f(n) = \Theta(g(n))$**

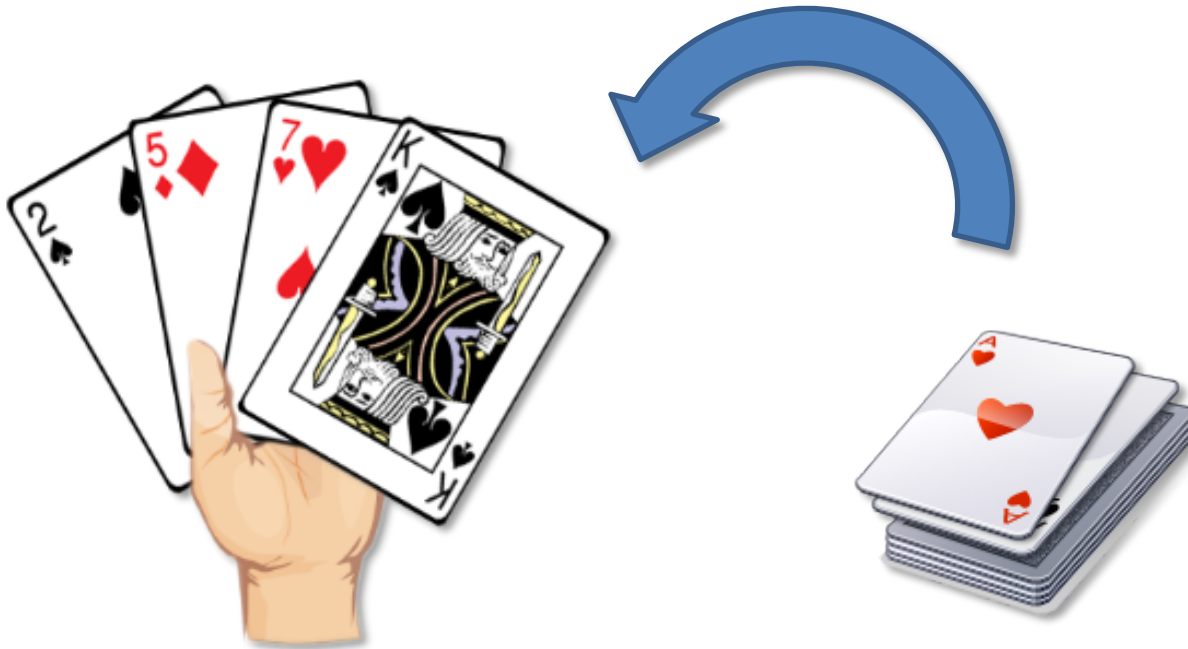
$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$  일 때  
e.g.  $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

※ 양의 상수  $n$ 과  $c$ , 계산 방법에 따라 여러 가지  $g(n)$ 을 구할 수 있다.  
단, 일반적으로 가장 근접한 값을 찾으려 한다.

# Insertion Sort

## ▶ Insertion Sort (삽입 정렬)

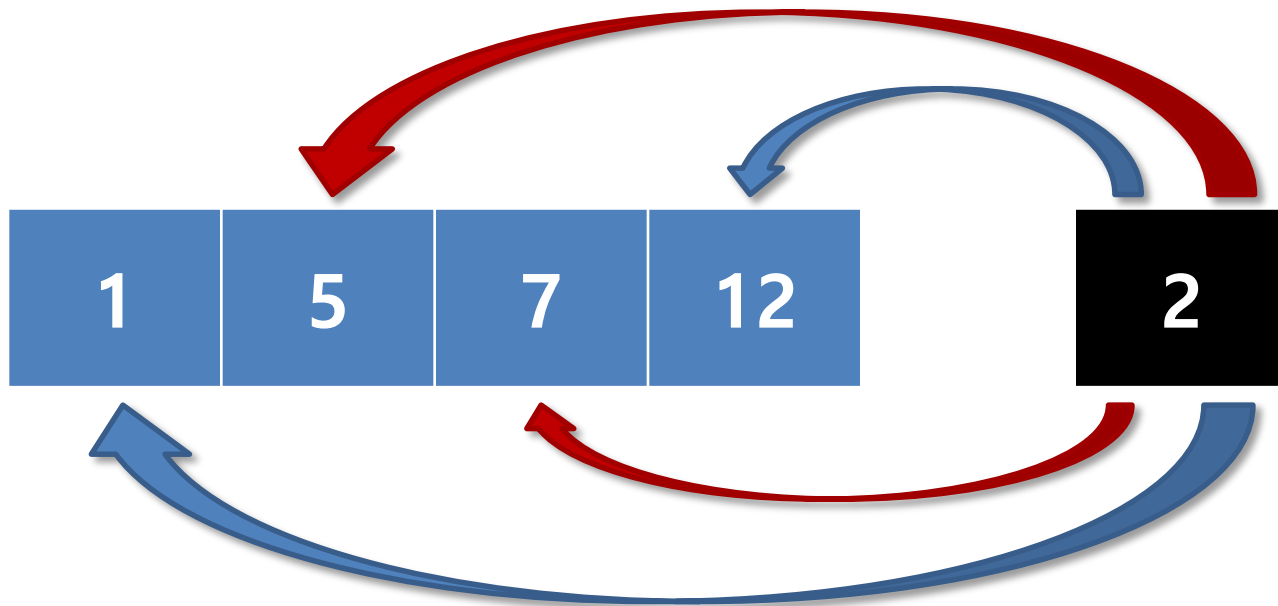
정렬되지 않은 배열로부터 **데이터를 하나씩 꺼내어**  
정렬되어 있는 배열의 알맞은 위치에 삽입하는 정렬 방법



# Insertion Sort

## ▶ 프로그램으로 구현 시 달라지는 점

알맞은 위치에 데이터를 삽입하기 위해  
배열에 저장된 값들을 하나씩 **순서대로 비교**해 보아야 함



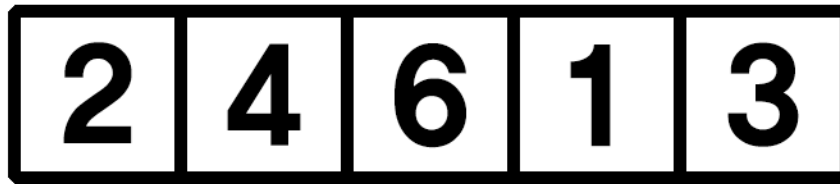
# Insertion Sort

## ▶ 정렬 방법 (1/7)

배열의 첫 번째 데이터를 정렬된 배열,  
나머지 데이터를 정렬되지 않은 배열로 나누어 생각한다



A[0]



A[1]

A[2]

A[3]

A[4]

A[5]

# Insertion Sort

## ▶ 정렬 방법 (2/7)

우측 배열의 첫 번째 데이터 값을 변수 **Key**에 복사한다



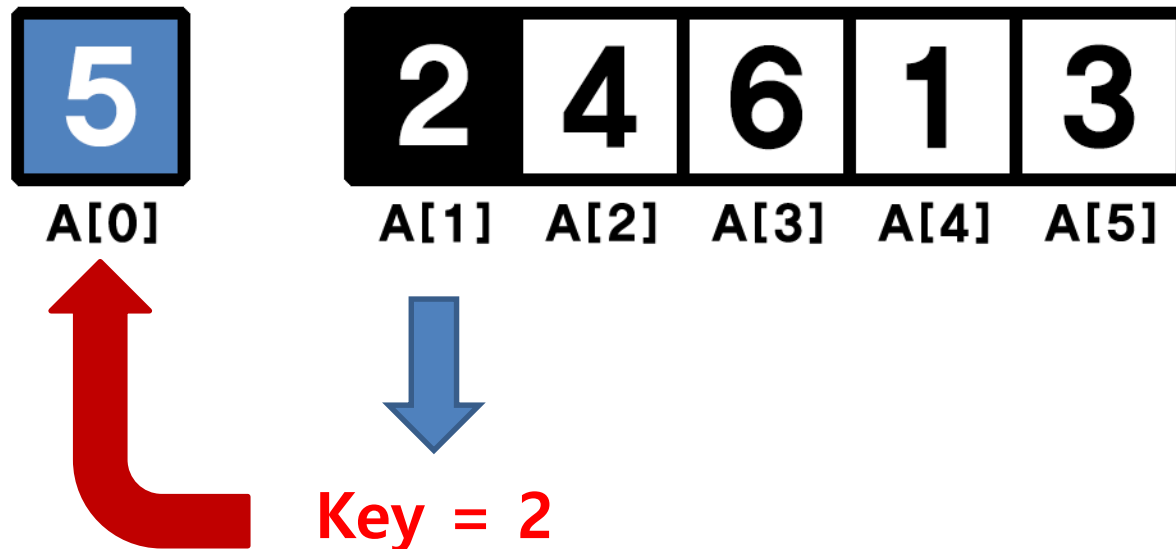
**Key = 2**



# Insertion Sort

## ▶ 정렬 방법 (3/7)

복사한 Key 값을 좌측 배열에 저장된 수 중  
가장 마지막(오른쪽) 배열에 저장된 값과 비교한다

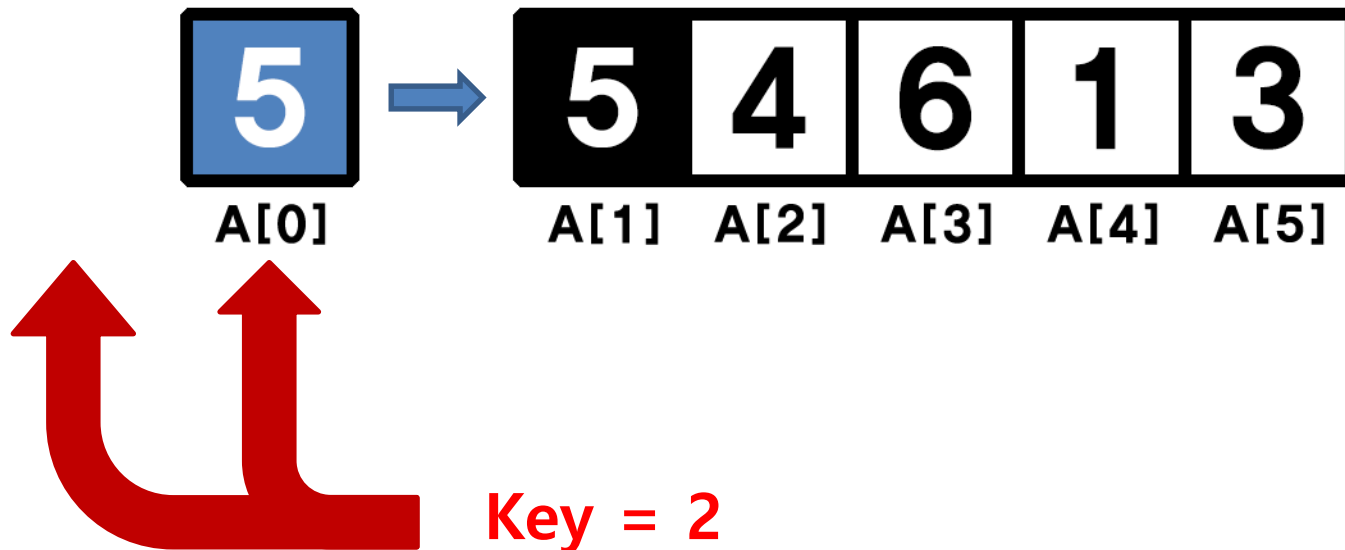


# Insertion Sort

## ▶ 정렬 방법 (4/7)

만약 비교한 값이 Key보다 크면

해당 값을 오른쪽 인덱스에 복사하고 그 왼쪽 값을 비교한다



# Insertion Sort

## ▶ 정렬 방법 (5/7)

더 비교할 값이 없거나, 비교한 값이 Key보다 작거나 같다면  
비교했던 위치의 바로 오른쪽 인덱스에 Key 값을 복사한다



# Insertion Sort

## ▶ 정렬 방법 (6/7)

1개 데이터에 대한 삽입 정렬이 완료되었다  
남은  $A[2] \sim A[5]$ 의 데이터에도 같은 작업을 반복한다



# Insertion Sort

## ▶ 정렬 방법 (7/7)

완료



# Insertion Sort

## ▶ pseudo-code (의사 코드)

“pseudocode” {

```
INSERTION-SORT ( $A, n$ )    ▷  $A[1 \dots n]$   
  for  $j \leftarrow 2$  to  $n$     ▷  $c_1 * n$   
    do  $key \leftarrow A[j]$     ▷  $c_2 * (n - 1)$   
       $i \leftarrow j - 1$     ▷  $c_3 * (n - 1)$   
        ▷  $c_4 * \sum_{j=2}^n t_j$  while  $i > 0$  and  $A[i] > key$   
          ▷  $c_5 * \sum_{j=2}^n (t_j - 1)$  do  $A[i+1] \leftarrow A[i]$   
            ▷  $c_6 * \sum_{j=2}^n (t_j - 1)$     $i \leftarrow i - 1$   
               $A[i+1] = key$     ▷  $c_7 * (n - 1)$ 
```

※ 반복문의 루프가 종료될 때, 한 번 더 검사를 수행하는 점에 유의한다.  
또한  $j = 2$  to  $n$  일 때의  $t_j$  값은, *best case* = 1, *worst case* =  $j$ 이다.

# Merge Sort

## ▶ Merge Sort (합병 정렬)

[분할] – [정복] – [결합] 과정을 **재귀적으로 반복**하는 정렬 방법

### (1) 분할 (Divide) → mergeSort()

배열의 크기가 1이 될 때까지 계속하여 **배열을 둘로 나눈다**

### (2) 정복 (Conquer) → merge()

나뉘진 데이터를 **2개 배열씩 비교하여 재귀적으로 정렬**한다

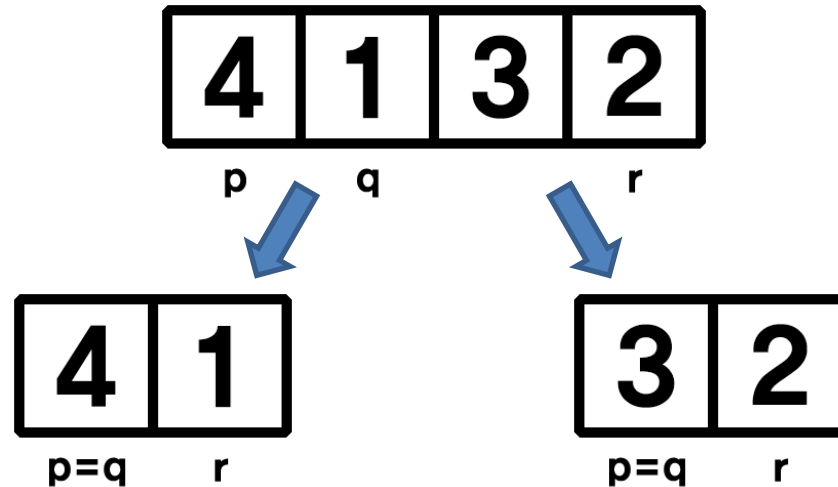
### (3) 결합 (Combine) → merge()

**정렬된 두 개의 배열을 병합**해 하나의 정렬된 배열로 만든다

# Merge Sort

## ▶ 정렬 방법 (1/4) – 분할 (1/2)

배열의 처음과 마지막 인덱스 번호를  $p$ ,  $r$ 이라 하고  
가운데 인덱스 번호를  $q$ 라 하여, 이를 기준으로 배열을 나눈다

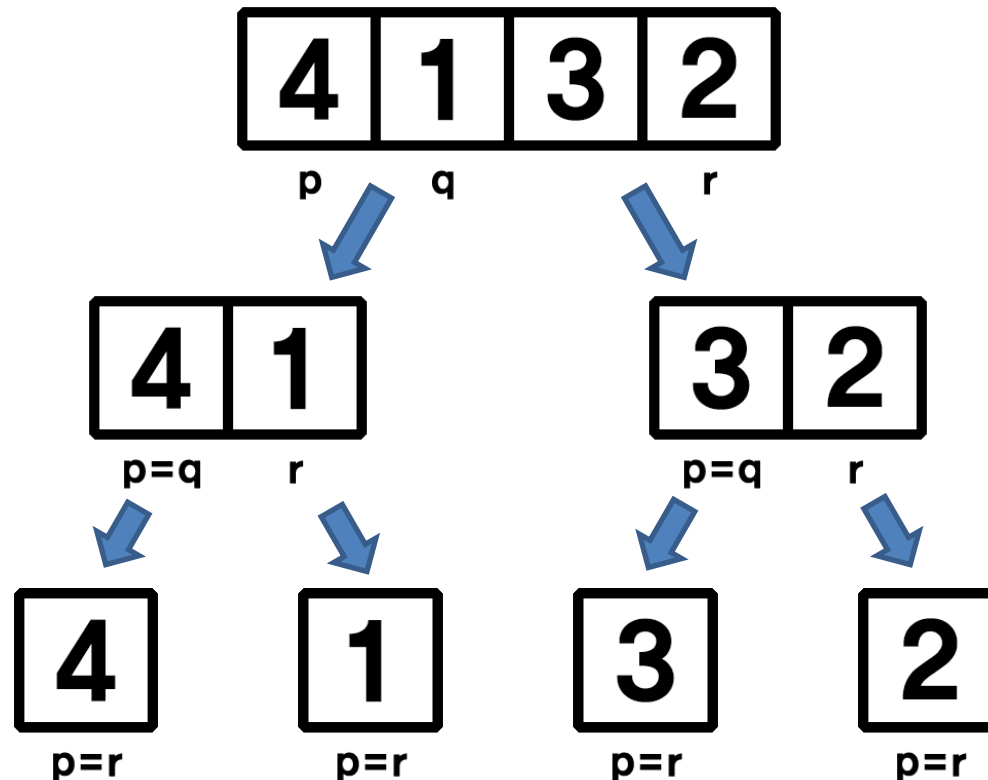




# Merge Sort

## ▶ 정렬 방법 (2/4) – 분할 (2/2)

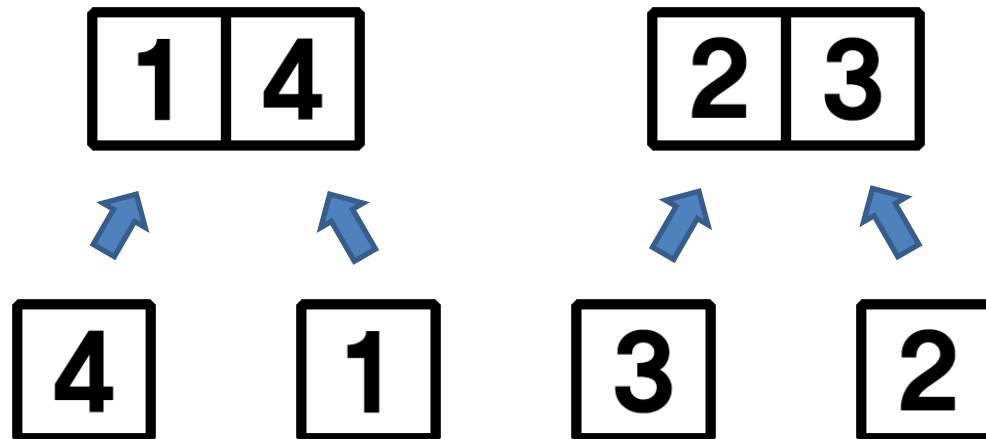
배열의 크기가 1이 될 때까지 계속하여 배열을 둘로 나눈다



# Merge Sort

## ▶ 정렬 방법 (3/4) – 정렬 & 결합 (1/2)

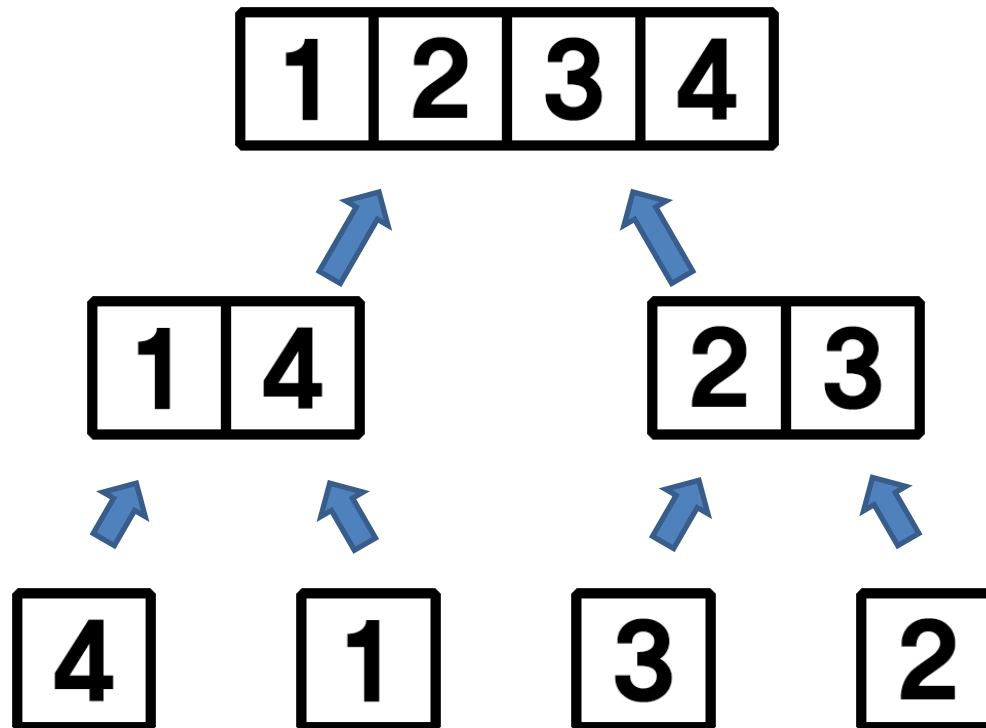
두 배열의 가장 앞 데이터를 비교하여  
더 작은 값부터 차례대로 뽑아내어 정렬 및 결합한다



# Merge Sort

## ▶ 정렬 방법 (4/4) – 정렬 & 결합 (2/2)

원래의 크기가 될 때까지 계속하여 정렬 및 결합하면 완료



# Merge Sort

**MERGE-SORT**  $A[1 \dots n]$

1. If  $n = 1$ , done.
2. Recursively sort  $A[1 \dots \lceil n/2 \rceil]$  and  $A[\lceil n/2 \rceil + 1 \dots n]$ .
3. “*Merge*” the 2 sorted lists.

*Key subroutine:* **MERGE**

# Practice / Homework

파일에서 숫자를 입력 받아 정렬 결과를 파일로 출력

1. Insertion Sort 구현
2. Merge Sort 구현
  - mergeSort()와 merge() 두 함수를 구현
  - mergeSort()는 Recursive Function으로 구현
3. Binary Insertion Sort 구현

요구사항

1. 프로그램 콘솔 창에는 정렬하는데 소요되는 시간만 출력
2. 보고서에는 1만~1000만까지 BEST, AVERAGE, WORST 경우에 소요되는 시간을 측정하여 그래프를 이용하여 비교 분석  
(테스트 크기가 컴퓨터 성능에 제한되는 경우에는 생략 가능)

주의사항

- a . 입력 파일 명 : "input.txt"
- b . 출력 파일 명 : "학번\_output.txt"(ex. 201701234\_output.txt)
- c . 정렬 후 출력된 파일은 Best폴더 안의 txt파일과 완전히 같은 파일이 되어야 함

# Practice / Homework

## ▶ 입력 파일 예시

각 숫자들은 공백 하나로 구분되어있음

```
10 7 6 5 1 3 4 2 8 9
```

## ▶ 출력 파일 예시

각 숫자 사이를 공백으로 구분하여 출력

마지막 숫자 뒤에는 공백을 넣지 말 것

```
1 2 3 4 5 6 7 8 9 10
```

# Practice / Homework

## ▶ 프로그램 구현 과정(예시)

경우에 따라 생략되는 과정이 있을 수 있음

#	구현 순서
1	파일 입/출력 변수 및 그 외 필요한 변수 선언
2	파일로부터 읽어온 데이터를 저장할 배열 생성
3	파일을 끝까지 읽어, 각각의 값을 배열에 저장
4	시간 측정 시작
5	<b>Sorting</b>
6	시간 측정 끝
7	정렬 결과를 파일로 출력
8	파일 닫기

# Practice / Homework

## ※ 그 외 실습 과제 수행 중 유의 사항 (JAVA)

- a . 사이버 캠퍼스에 과제 제출, 구현한 소스파일(.java)만 zip으로 압축하여 보낼 것. (폴더로 압축하지 않도록)
- b . 과제 평가는 별도의 input data를 사용함. (양식은 동일)



# Practice / Homework

과제 제출 안내	
제출 파일	각 정렬 코드 (3개) 보고서 파일 (1개)
제출 기한	9월 21일 (목) <b>실습 수업 시간 전까지</b>

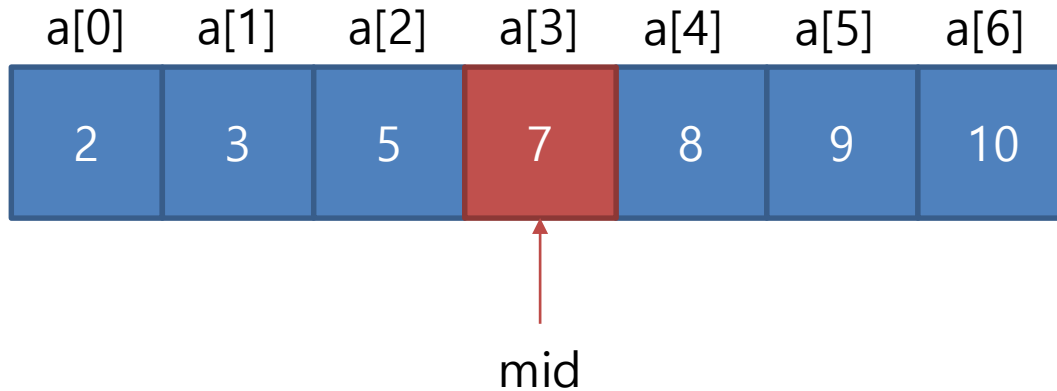
과제 평가 감점 사항	
제출 지연 (수업 시작부터)	- 50% / 1주
요구 사항 누락 / 결과값 불일치	- 10 ~ 20% / 1개
코드 Error	- 50 ~ 100%
과제 Copy	0점

# hint. binary search

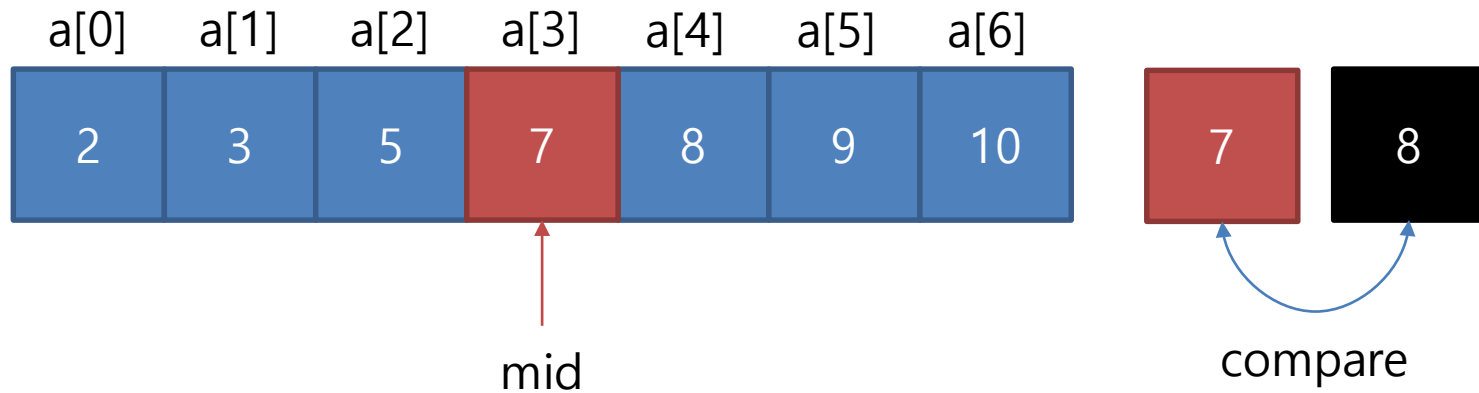
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
2	3	5	7	8	9	10

8

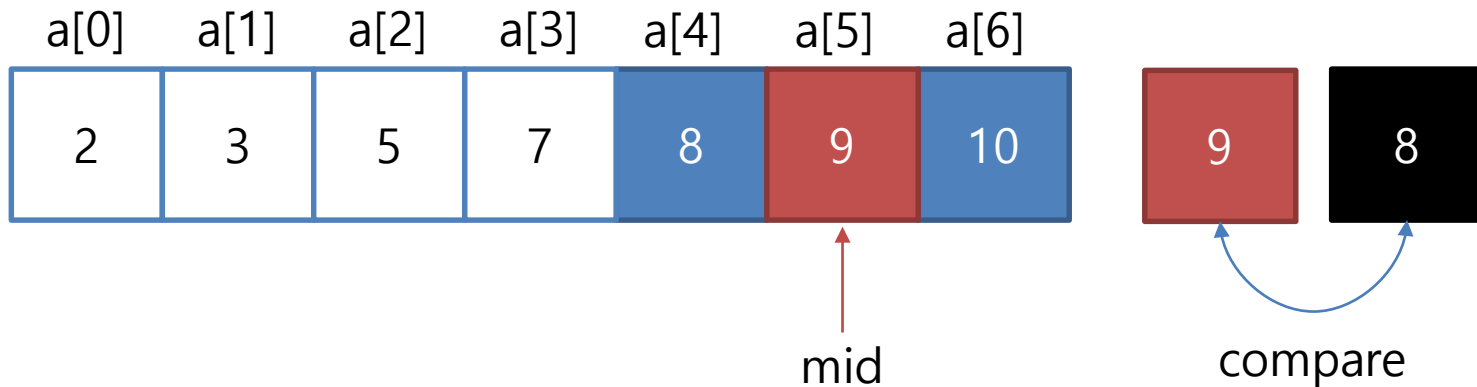
# hint. binary search



# hint. binary search



# hint. binary search



# hint. binary search

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
2	3	5	7	8	9	10

mid



compare