Algorithms

Kiho Choi

Fall, 2022

Department of AI·Software
Gachon University

Course Orientation I

- Text books
  - Introduction to Algorithms, 3rd edition by Cormen, Leiserson, Rivest, and Stein, The MIT Press, 2009
  - Programming challenges, by Steven S. Skiena, Miguel A. Revilla, Springer, 2003

- Instructor
  - Prof. Kiho Choi (Department of AI·Software)
    - E-mail: aikiho@gachon.ac.kr
    - Office : AI Building 417

Course Orientation II

- Grading Scheme
  - Midterm examination: 25% ,
  - Final examination:25%
  - Quiz, Homework , term project : 35%
  - attendance: 15%

Course Description

- Active learning and MOOC class will be decided later.

Wk	Topic	Etc.
1	basics of algorithm design and analysis	
2	growth of functions, divide-and-conquer	
3	dynamic programming I	
4	dynamic programming II	
5	greedy algorithms I /greedy algorithms II	
6	graph algorithms I	
7	graph algorithms II	
8	midterm examination	
9	number-theoretic algorithms	
10	computational geometry I	
11	computational geometry II	
12	backtracking I	
13	backtracking II	
14	approximation algorithms, NP-Completeness	
15	final examination	

Course Description

- This course teaches techniques for the design and analysis of efficient algorithms, emphasizing methods useful in practice.
- Students will learn a number of important basic algorithms.
- Students who successfully complete this course will be able to analyze and design efficient algorithms for a variety of computational problems.
- They will be also be able to communicate their ideas in the form of precise algorithm descriptions.

Course Objectives

Course Rules

- Camera should be “on” in the online class
- Make-up/late homework will not be graded for credit.
- Cheating in exams and quizzes will receive an “F” for the course
- “not attending” 4 or more classes will result in course grade below “F”
- “not attending” a class includes
  - not attending a class
  - being late to a class
  - leaving a class in the middle
  - chatting in class
  - having the mobile phone on in class

What's algorithm?

News

삼성 채용 변화

기존
- 서류전형(학점/어학점수)
- 직무적성검사(SSAT)
- 실무면접
- 임원면접

→ **2015 하반기**
- 서류전형(**직무적합성평가)**
- 직무적성검사(SSAT)
- 실무면접
- **창의성 면접**
- 임원면접

→ **2016~**

삼성 서류전형(직무적합성 평가)

영업/경영지원직(인문계)
- 직무적합성 : 직무에세이

연구개발/기술직(이공계)
- 전공능력
- 전공 이수과목 및 성적

소프웨어
- SSAT 대신 소프트웨어 역량테스트 (코딩 및 알고리즘 개발 능력 평가)

각 부문별 직무적합성 평가 및 SSAT 적용 내용

소프트웨어직 : SSAT 대신 소프트웨어 역량테스트(코딩 및 알고리즘 개발 능력 평가)를 신설하여 평가할 예정이라고 한다. https://swexpertacademy.com/main/main.do

1. Basics of Algorithm Design and Analysis

Contents

- What’s an Algorithm?
- Objectives of Studying Algorithm
- Desirable Algorithm

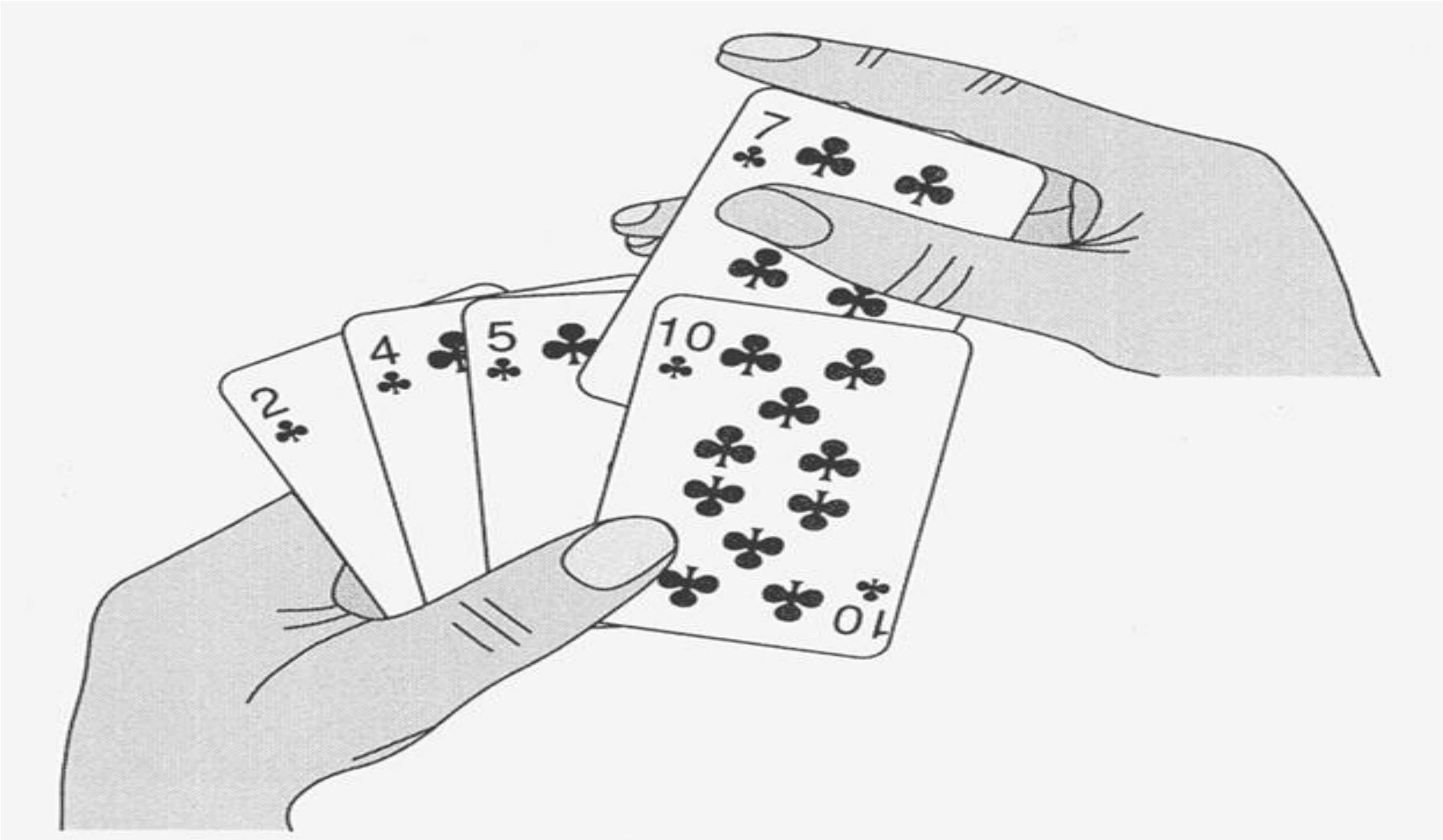
What is an algorithm?

- An algorithm is a finite set of instructions that if followed accomplishes a particular task.
  - *well-defined computational procedure* that tasks some value, or set of values, as input and produces some value, or set of values, as output.
  - a sequence of computational steps that transform the input into the output.
  - A tool for solving a well-specified computational problem.

The Problem of sorting

- Example Input/Output
  - Input
    - A sequence of n numbers $<a_1, a_2, \ldots, a_n>$
  - Output
    - A permutation (reordering) $<a_1, a_2, \ldots, a_n>$ of the input sequence such that $a_1<a_2< \ldots<a_n$.
  - Example
    - 8, 2, 4, 9, 3, 6
    - 2, 3, 4, 6, 8, 9

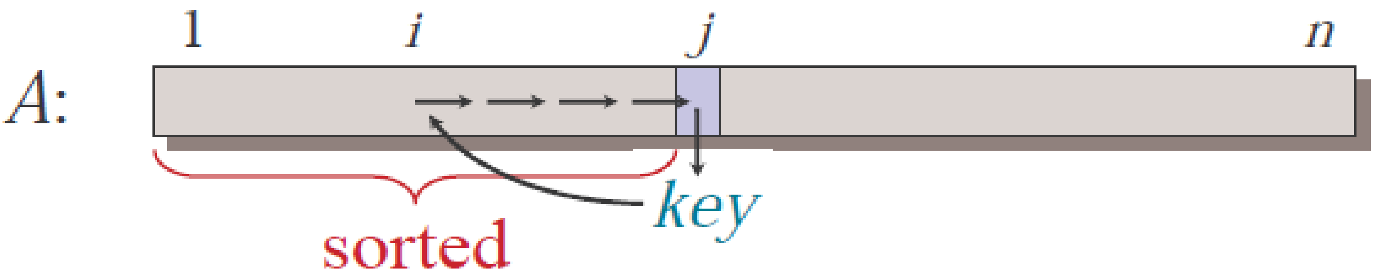
Examples of insertion sort



Examples of insertion sort

“pseudocode”

```
INSERTION-SORT (A, n)      ◁ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```



https://www.youtube.com/watch?v=OGzPmgsI-pQ

Examples of insertion sort

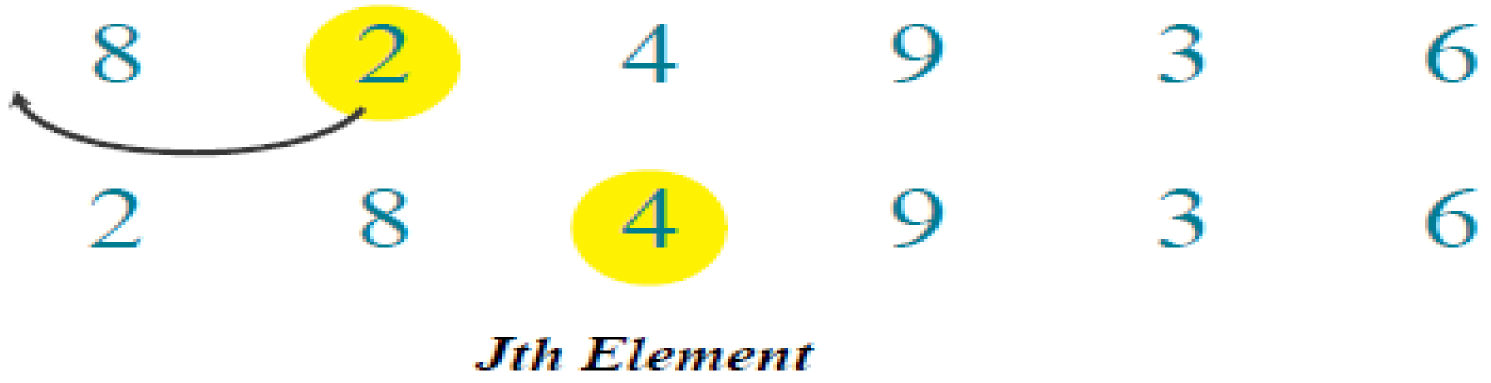
8 2 4 9 3 6

Jth Element

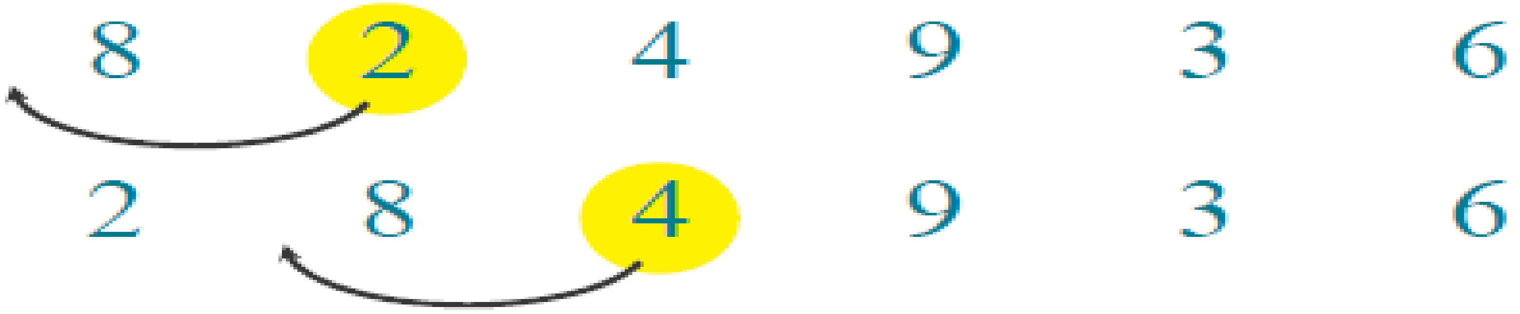
Examples of insertion sort

8 2 4 9 3 6

Examples of insertion sort



Examples of insertion sort

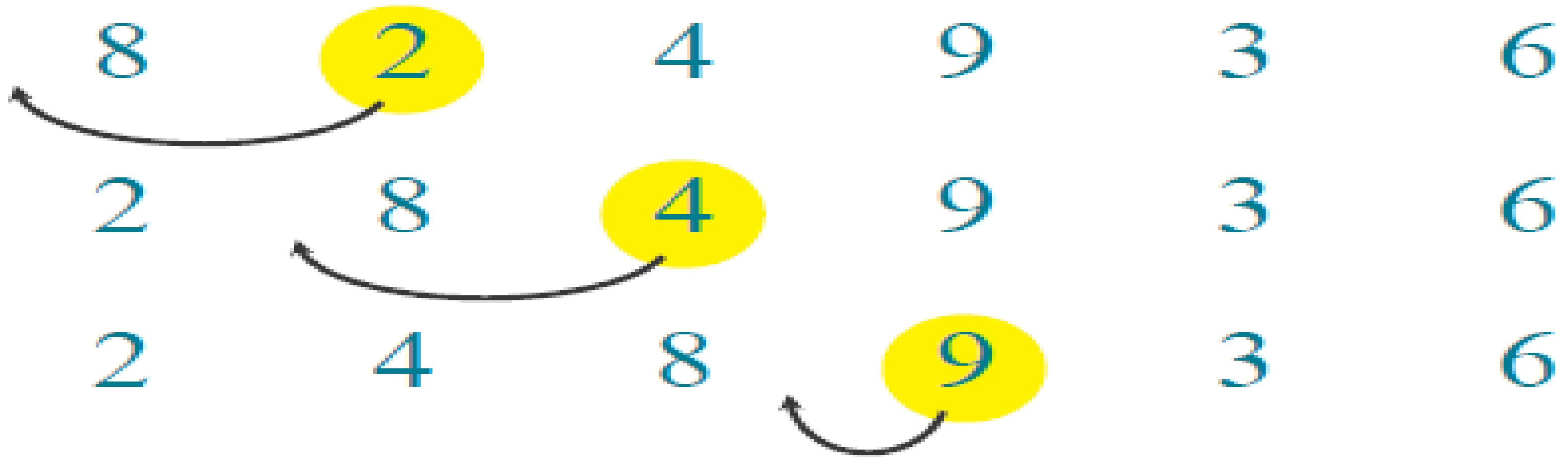


Examples of insertion sort

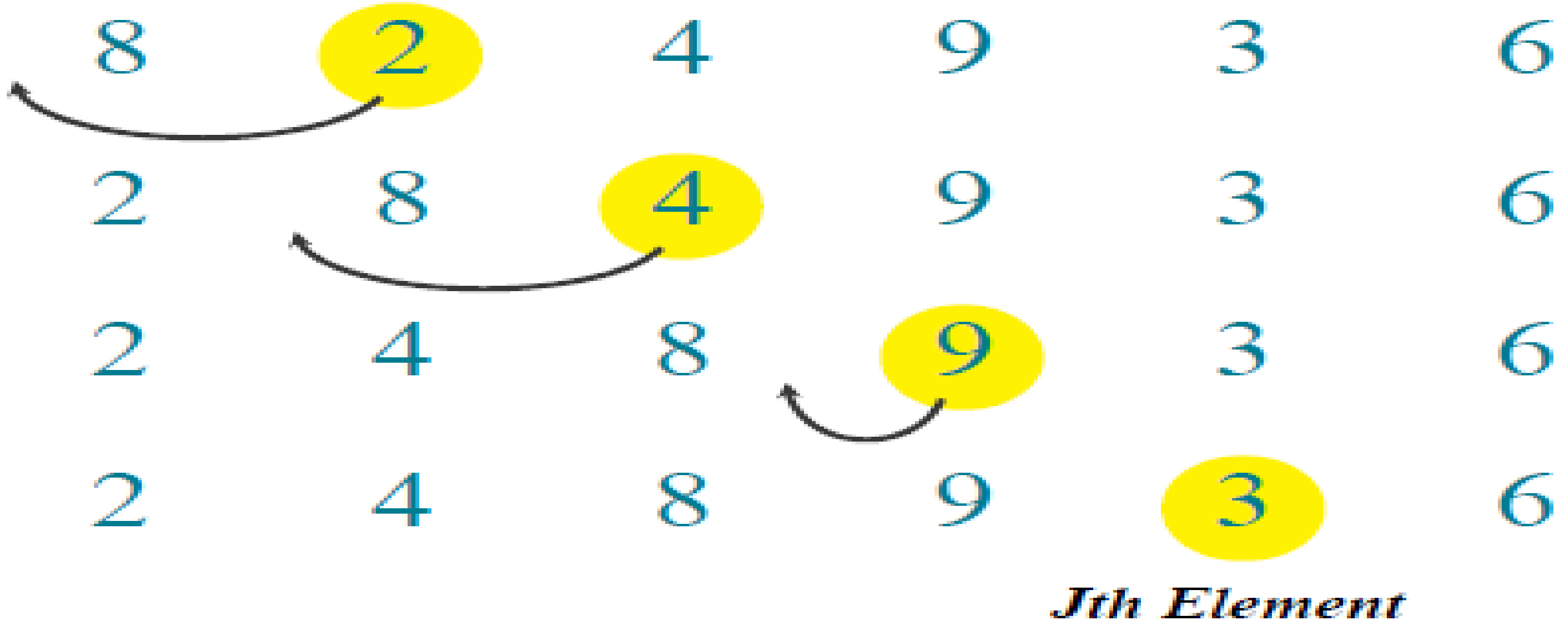
8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6

Jth Element

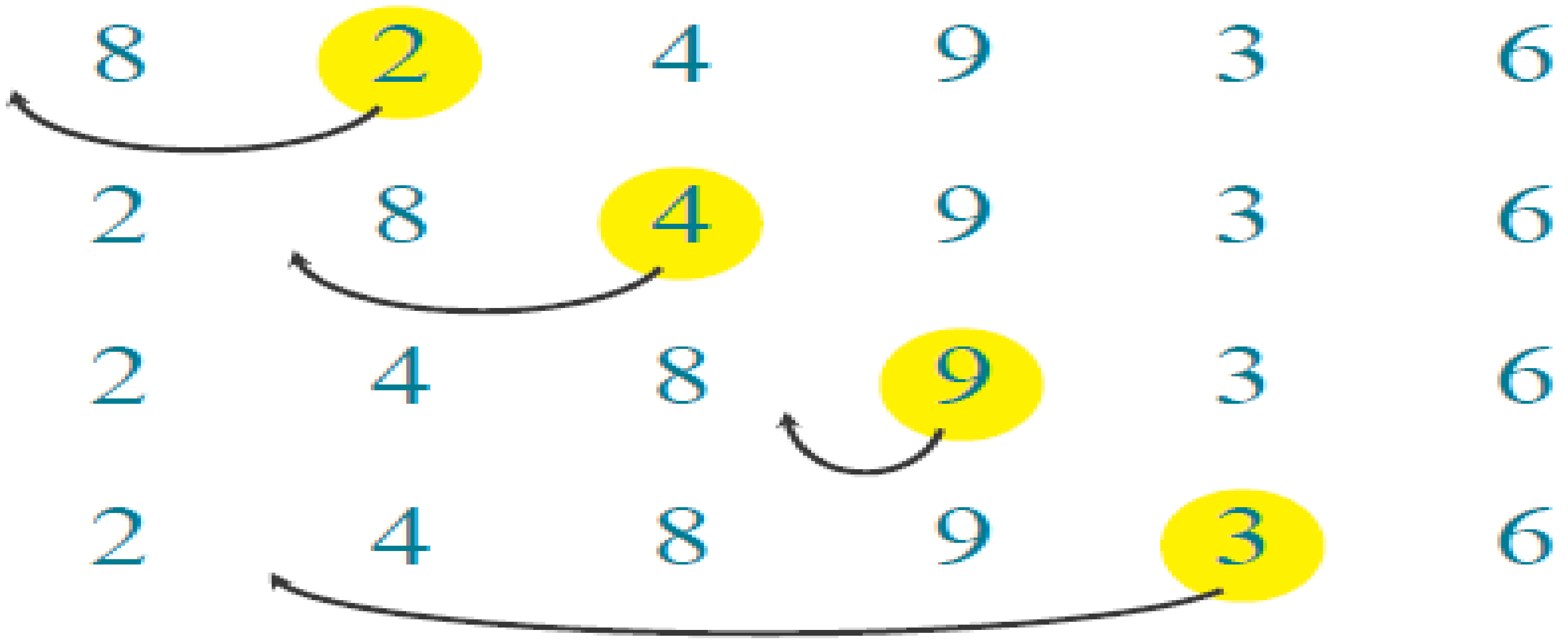
Examples of insertion sort



Examples of insertion sort



Examples of insertion sort



Examples of insertion sort

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6
2	4	8	9	3	6
2	3	4	8	9	6

Jth Element

Examples of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

Examples of insertion sort

8	2	4	9	3	6	
2	8	4	9	3	6	
2	4	8	9	3	6	
2	4	8	9	3	6	
2	3	4	8	9	6	
2	3	4	6	8	9	*done*

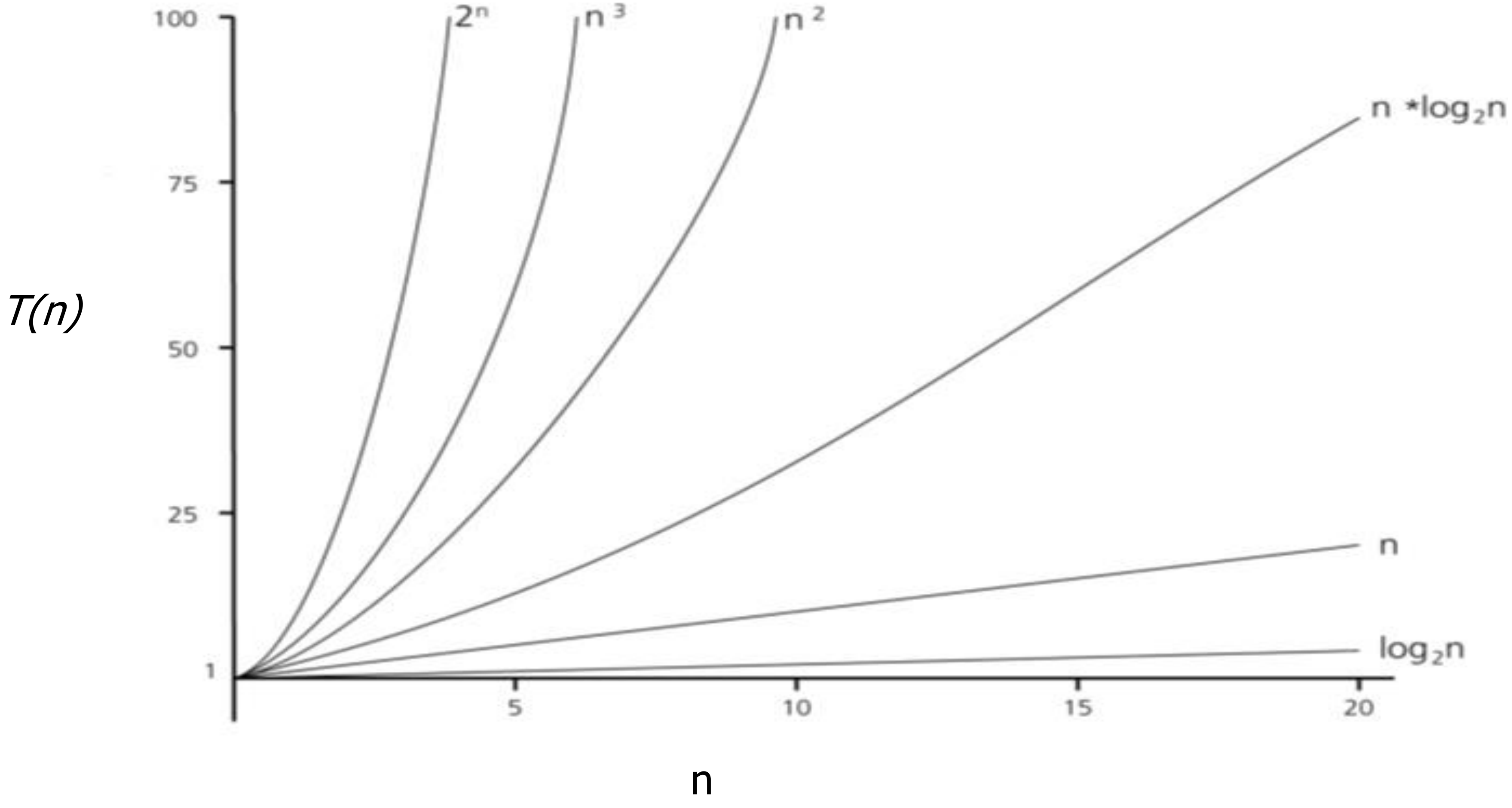
Desirable Algorithm

- An algorithm must satisfy the following criteria.
  - **Input**: zero or more inputs are supplied.
  - **Output**: at least one output should be produced as results of procedure.
  - **Definiteness**: each instruction should be clear and unambiguous (i.e.) not more than one meaning.
  - **Finiteness**: if all the instructions are traced in algorithm, then for all cases the algorithm must terminate after a finite number of steps.
  - **Effectiveness**: Every instruction must be very basic so that it can be carried out briefly said "Operation must be feasible".

Running time

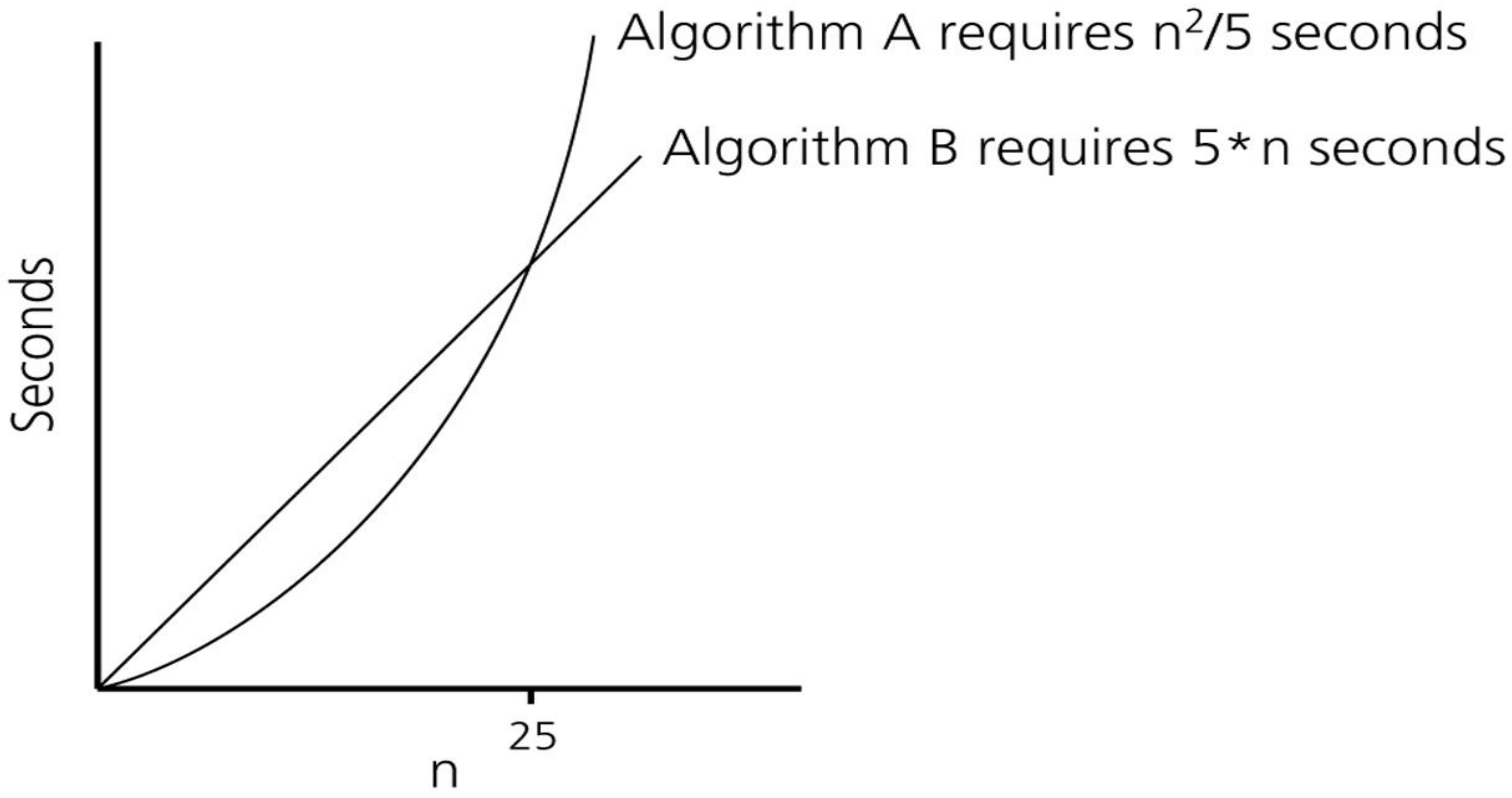
- How do we analyze an algorithm’s running time?
- The time taken by an algorithm depends on the input
  - Sorting 100 numbers takes longer than sorting 3 numbers.
  - A given sorting algorithm may even take differing amounts of time on two inputs of the same size.
  - For example, we’ll see that insertion sort takes less time to sort n elements when they are already sorted than when they are in reverse sorted order.

Running time



Running time

$T(n)$



Running time

Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Running time

```
sample1(A[ ], n)
{
        k = n/2 ;
        return A[k] ;
}
```

Running time

```
sample2(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                sum← sum+ A[i] ;
        return sum ;
}
```

Running time

```
sample3(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                for j ← 1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

Running time

```
sample4(A[ ], n)
{
    sum ← 0 ;
    for i ← 1 to n
        for j ← 1 to n {
            k ← Max A[1 ... n] ;
            sum ← sum + k ;
        }
    return sum ;
}
```

Running time

```
sample5(A[ ], n)
{
        sum ← 0 ;
        for  i ← 1 to n
                for  j ←  i+1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

Kinds of analyses

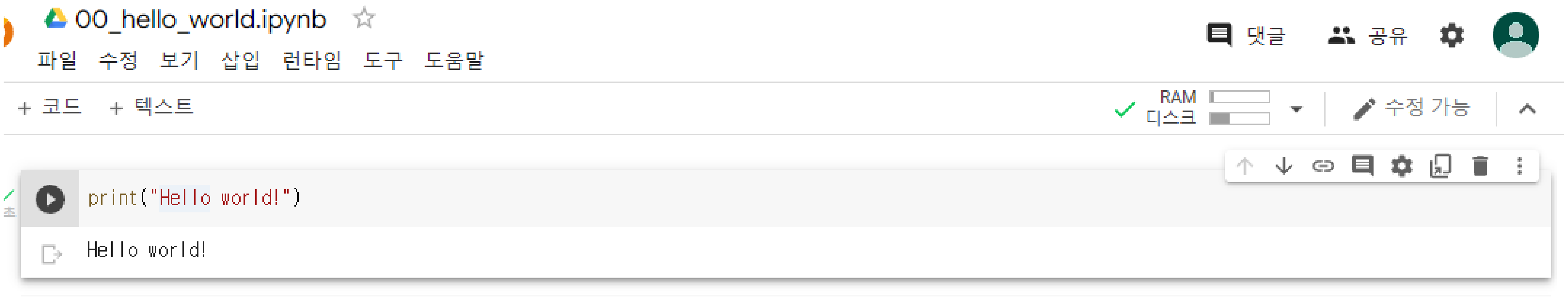
- Worst-case (usually)
  - T(n) = maximum time of algorithm on any input of size n.
- Average-case (sometimes)
  - T(n) = expected time of algorithm over all inputs of size n.
  - Need assumption of statistical distribution of inputs
- Best-case (bogus)
  - Cheat with a slow algorithm that works fast on some input

Implementation

Python testing environment

- Using Colab for python programming
- Please refer the following link for the setup:
  - https://theorydb.github.io/dev/2019/08/23/dev-ml-colab/

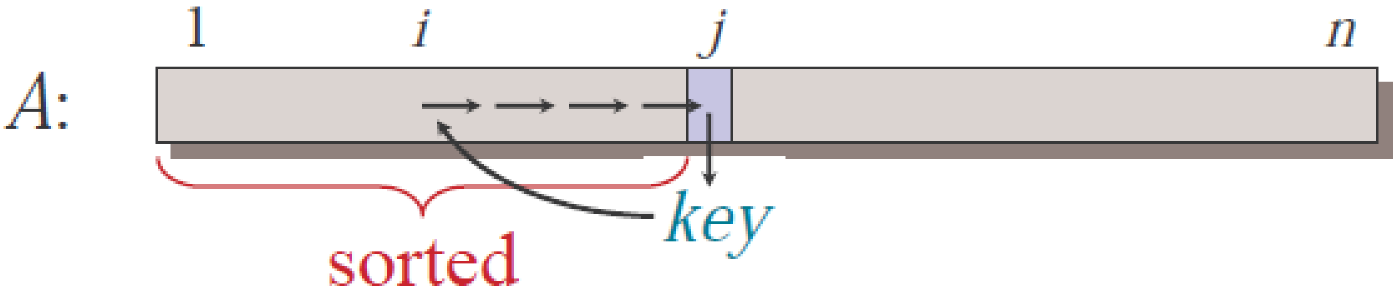
- Example code



Implementation of insertion sort

"pseudocode"

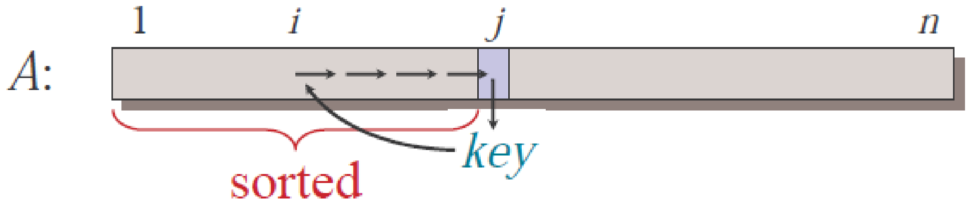
```
INSERTION-SORT (A, n)      ◁ A[1 . . n]
    for j ← 2 to n
          do key ← A[j]
             i ← j – 1
             while i > 0 and A[i] > key
                   do A[i+1] ← A[i]
                      i ← i – 1
             A[i+1] = key
```



Implementation of insertion sort

"pseudocode"

```
INSERTION-SORT (A, n)        ◁ A[1 . . n]
    for j ← 2 to n
         do key ← A[ j]
            i ← j – 1
            while i > 0 and A[i] > key
                 do A[i+1] ← A[i]
                    i ← i – 1
            A[i+1] = key
```



Implementaion of insertion sort

```python
[1]  # Definition of insertion sort
     def insertionSort(A):
       for j in range(1, len(A)):
         key = A[j]
         i = j-1
         while i >= 0 and A[i] > key:
           A[i + 1] = A[i]
           i -= 1
         A[i+1] = key
```

```python
[2]  # List
     input_list1 = [8, 2, 4, 9, 3, 6]
     print(input_list1)
```

```
     [8, 2, 4, 9, 3, 6]
```

```python
[3]  # Sorting
     insertionSort(input_list1)
     print(input_list1)
```

```
     [2, 3, 4, 6, 8, 9]
```

Random list generation

```python
[4]  # random list
     import random
     input_list2 = random.sample(range(100),10)
     print(input_list2)
```

```
     [66, 68, 25, 88, 86, 8, 99, 63, 82, 77]
```

```python
[5]  # Sorting
     insertionSort(input_list2)
     print(input_list2)
```

```
     [8, 25, 63, 66, 68, 77, 82, 86, 88, 99]
```

Example code test

- Code test: https://www.acmicpc.net/problem/2750
- Solving the problem using insertion sort
- Example result of submission

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
48321793	aikiho	2750	맞았습니다!!	30840 KB	208 ms	Python 3 / 수정	430 B	4분 전

THANK YOU