Lecture #0
Introduction

Algorithm

JBNU

Jinhong Jung

Outline

- ❑ Course Overview
- ❑ Things We’ll Learn
- ❑ Nagging Advices

Course Information

Instructor

- **Name:** Jinhong Jung [Homepage]
- **E-mail:** jinhongjung@jbnu.ac.kr

Online Tools

- CLASSUM: feel free to ask any questions anonymously
  - https://www.classum.com/HKDKFRMH7
- LMS: score notifications, attendance check, etc.
  - https://ieilms.jbnu.ac.kr

Reference Text

❑ 쉽게 배우는 알고리즘 (개정판)

- 저자: 문병로
- 출간: 2018
- 출판사: 한빛 아카데미

❑ See the textbook

- For reviewing what you learned from each lecture
- For studying details of an algorithm and its related proofs
- For solving exercise problems

Prerequisites

- Data structure (✶✶✶)
  - Able to implement commonly used data structures
  - Able to analyze their time & space complexities
- C++ Programming (✶✶)
  - Able to implement a pseudocode in C++
  - Able to analyze, debug, and fix your C++ codes by yourself
- Discrete mathematics (✶)
  - Familiar with mathematical expressions
  - Useful when you try to prove a claim logically

Logistics

❑ Teaching environment

- Lectures are offered via Zoom, but **exams are taken offline**

❑ Teaching language

- Korean is used by default in all aspects **except for lecture slides** written in **English**

❑ Q&A policies

- While streaming a lecture, ask your questions via Zoom chat
- After the lecture, ask your questions related to lecture content via **CLASSUM**
    - Questions on lecture content via e-mail will be rejected
    - You can use e-mail to inquiry your personal excuse (e.g., 공결처리)

Evaluation Plan (1)

❑ Grading policy: relative evaluation (상대평가)

- The ratio for Grades A and B should be 80%

Grade A	Grade B	Grade C	Others
40%	40%	15%	5%

❑ Proportion for grading

Attendance	Homework	Midterm Exam	Final Exam
5%	25%	35%	35%

- The maximum total score is 100 points

Evaluation Plan (2)

Deadline and disqualification policies

- **No plan to accept any submission after its deadline**
  - Check LMS regularly and do your homework in advance!
- Any form of **cheating** will give the corresponding task of its contributors **a zero point**
  - ⚠️ It could be reported to **our school's disciplinary committee**
- If you have absences of **more than a quarter (1/4) of all lecture time**, then **Grade F** will be assigned immediately
  - This course will take 45 hours in this semester
  - Thus, the absence of 11.25 hours presents Grade F to you
- If your total score is **less than 10 points**, then **Grade F** will be assigned immediately

FAQ on Programming Assignment (1)

- 🙋 **I didn't keep the deadline for some reasons, so can I submit it right now?**
  - 📢 **No!** Check the schedule regularly, and do your homework in advance!

- 🙋 **Can I use STL in C++ for PA?**
  - 📢 **Partly allowed!** You can use data structures in STL, but are not allowed to use some functions in STL related to algorithms what you learned in this course.

FAQ on Programming Assignment (2)

🙋 **My code works well in MS Visual Studio, but a compile error occurs when it's submitted to DMOJ**

- 📢 DMOJ uses GNU g++ compiler which couldn't be compatible with MSVC in terms of API [link]
  - ➢ **Use the standard compiler always**! If you're a Window user, see [link] to set up GNU G++

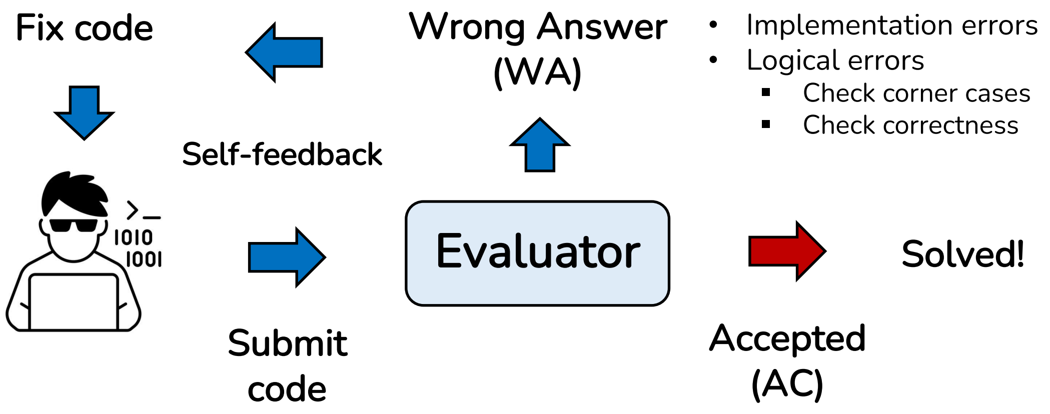
🙋 **Can you give me input/output cases used for grading?**

- 📢 **No!** It's like asking for correct answers in exams.

FAQ on Programming Assignment (3)

🙋 I got Wrong Answer (WA) while my code works correctly for sample cases. Could you tell me where I'm wrong in my code?

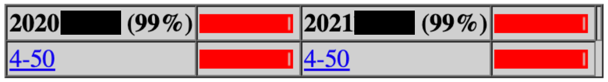
- 📢 No! MUST resolve WA by yourself, which is the original intention of PA
  - ➢ You should first suspect your code and logic when WA appears!
  - ➢ CLASSUM posts containing code on PA will be deleted immediately



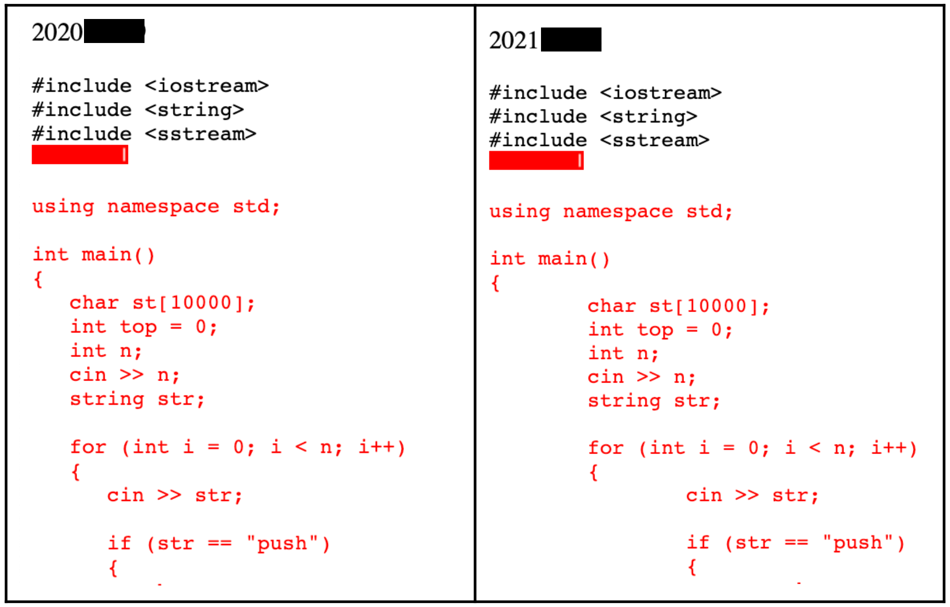
FAQ on Programming Assignment (4)

🙋 Never plagiarize code! Givers and plagiarists will get zero points

- 📢 Do your homework by yourself!



2020[redacted] (99%)	🟥	2021[redacted] (99%)	🟥
4-50	🟥	4-50	🟥



2020[redacted]

```
#include <iostream>
#include <string>
#include <sstream>
[redacted]

using namespace std;

int main()
{
   char st[10000];
   int top = 0;
   int n;
   cin >> n;
   string str;

   for (int i = 0; i < n; i++)
   {
      cin >> str;

      if (str == "push")
      {
```

2021[redacted]

```
#include <iostream>
#include <string>
#include <sstream>
[redacted]

using namespace std;

int main()
{
        char st[10000];
        int top = 0;
        int n;
        cin >> n;
        string str;

        for (int i = 0; i < n; i++)
        {
                cin >> str;

                if (str == "push")
                {
```

Outline

- ❑ Course Overview
- ❑ Things We’ll Learn
- ❑ Nagging Advices

Purpose Of This Course

- **Goal 1) Correctness and efficiency**
  - To understand how to design and analyze **an algorithm** in terms of correctness and efficiency

- **Goal 2) Classical algorithms**
  - To learn **algorithmic** ideas that effectively resolve challenges behind various classical problems

- **Goal 3) Computational thinking**
  - To improve your computational thinking that is used for solving new problems

What Is Algorithm?

- Sequential process for solving a problem using a computer(s) from the input data to the final output
  - When we check or design an algorithm, must clarify its inputs and outputs first!

Input

1/2 cup Butter
1/2 cup packed Sugar
1 Egg
1 1/2 cup Flour
1/2 tsp Baking Soda
1 1/2 tsp Ginger
1 tsp Cinnamon

Algorithm

Step 1. Combine flour, baking soda, ginger and cinnamon and set aside.
Step 2. Cream butterscotch pudding mix, butter and brown sugar.
Step 3. Add egg and mix.
Step 4. Add in flour mixture and mix ..
… …

Output

Very Simple Example

- Problem: Find a maximum of 100 students' scores
  - Step 1) Clarify the input and output
    - What is the input? ⇒ (Informal) 100 scores
      - (Formal) An array $\mathbf{x}$ of size 100, containing the input scores (i.e., $\mathbf{x}[1], \cdots, \mathbf{x}[100]$)
    - What is the output? ⇒ (Informal) A maximum of the input scores
      - (Formal) A maximum value among $\mathbf{x}[1], \cdots, \mathbf{x}[100]$
  - Step 2) Describe an algorithm for solving the problem
    - Given the input, it should correctly produce the output.

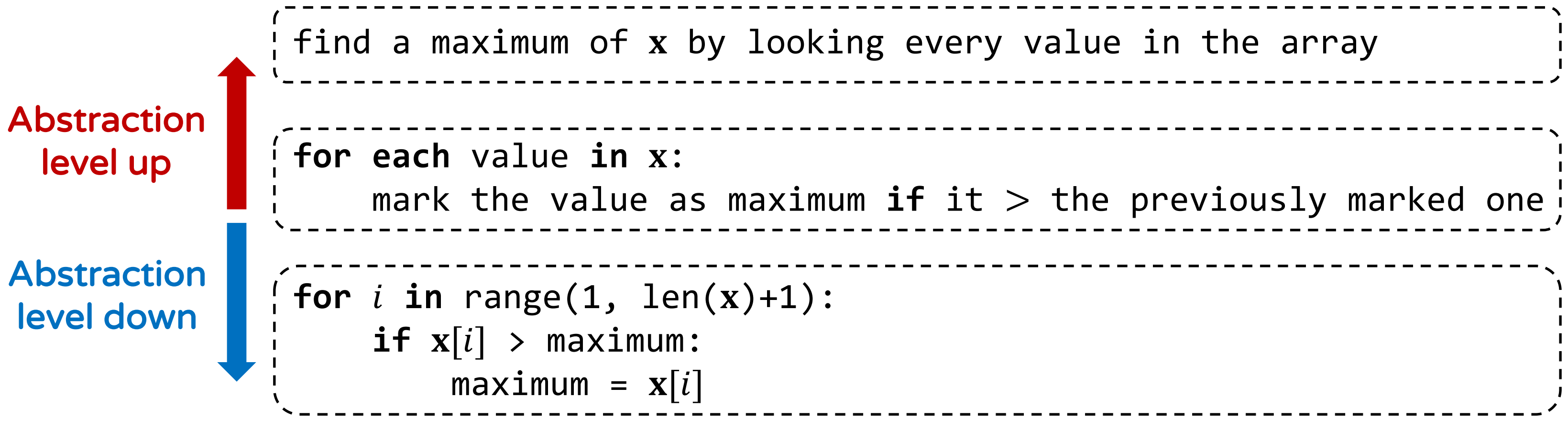
```
def max_score(x):

    for each value in x:
        mark the value as maximum if it > the previously marked one

    return the last marked value
```

Algorithm Should Be…

- ❑ Efficient and correctly designed
  - Practically efficient for large size of input using a computer
  - Must produce correct answers of the target problem
- ❑ Easy-to-understand
  - Easily able to implement your algorithm described in a pseudocode
  - Need to consider the abstraction level of target readers



```
find a maximum of x by looking every value in the array
```

```
for each value in x:
    mark the value as maximum if it > the previously marked one
```

```
for $i$ in range(1, len(x)+1):
    if x[$i$] > maximum:
        maximum = x[$i$]
```

How To Analyze Algorithms

- **You'll learn mathematical tools for analyzing algorithms w.r.t. correctness and efficiency**
  - Computational complexity
  - Asymptotic analysis
    - Big-O, Big-Omega, and Big-Theta notations
  - Recursion
  - Mathematical induction
  - Contradiction
  - Master theorem

Classical Problems (1)

- You'll learn ideas behind algorithms that solves various classical and basic problems.
  - Sorting algorithms
    - Efficiently rearrange elements of an array in a certain order
      - Selection, bubble, and insertion sorts
      - Merge, quick, and heap sorts
      - Count and radix sorts
  - Selection algorithms
    - Quickly find i-th smallest number in an unsorted array

Classical Problems (2)

- You'll learn ideas behind algorithms that solves various classical and basic problems.
  - Advanced data structures – disjoint set
    - How to efficiently manage multiple non-overlapping sets?
  - Graph algorithms
    - Minimum spanning tree
    - Single source shortest paths
    - Topological sort
  - String matching algorithms
    - How to efficiently find a word in a document?

Algorithmic Strategies & NP

- **You'll learn algorithmic strategies commonly used for designing algorithms to solve a problem**
  - Brute-force method
  - Divide and conquer
  - Dynamic programming
  - Greedy strategy
  - State space search
- **Also, you'll learn concepts of NP!**
  - To understand why some problems cannot be solved efficiently at least for now
    - NP, polynomial-time reduction, NP-Hard, NP-Complete

Schedule (Tentative)

Week	Content	Week	Content
1	Introduction Algorithm Analysis (1)	9	Graph Algorithms (1) Graph Algorithms (2)
2	Algorithm Analysis (2) Recursion (1)	10	Graph Algorithms (3) Graph Algorithms (4)
3	Recursion (2) Sorting Algorithms (1)	11	String Matching (1) String Matching (2)
4	Sorting Algorithms (2) Sorting Algorithms (3)	12	String Matching (3) NP (1)
5	Sorting Algorithms (4) Selection Algorithm	13	NP (2) NP (3)
6	Dynamic Programming (1) Dynamic Programming (2)	14	State Space Search (1) State Space Search (2)
7	Greedy Algorithms Disjoint Set	15	Final Exam
8	Midterm Exam		

⚠️ No plan to review programming skills and details of basic data structures in this schedule!

Outline

❑ Course Overview

❑ Things We’ll Learn

❑ Nagging Advices

Level of Difficulty

- This course aims at covering theoretical topics and fostering professional programmers in CSE
  - There are many topics to learn (see the schedule)
  - Several topics are not straightforward to understand
    - Should understand ideas, examples, proofs with math, pseudocode, and implementation
  - You may feel this course is fast-paced unless you aren't hardworking
    - Listen carefully to what I'm saying in each lecture
    - Review each lecture and resolve any questions you may have
    - Explain what you learn to your friends and solve exercises
    - If not, you may feel severe pain and give up following after midterm 😢

Still, I'm Not Sure...

- Answer the below questions within 5 min.
  - **Q1.** What do the below pseudocode mean?
  - **Q2.** Can I translate it into C++ in my head?

```
def func(s):
    visited[s] ← true
    queue.enqueue(s)

    while queue is not empty:
        u ← queue.dequeue()

        for each v in neighbors of u:
            if visited[v] is false:
                visited[v] ← true
                queue.enqueue(v)
```

Cases For Self-diagnosis

- Case 1) I can answer both questions immediately
  - ⇒ Don’t worry about it! Go straight!

- Case 2) I know what it’s for, but can’t implement
  - You will have trouble doing programming assignments
    - ⇒ Need to practice C++ programming for data structures

- Case 3) I can’t answer both questions
  - You are NOT going to follow well this course
    - ⇒ Need to study data structure by yourself if you must take this course

If You’re In Cases 2 & 3

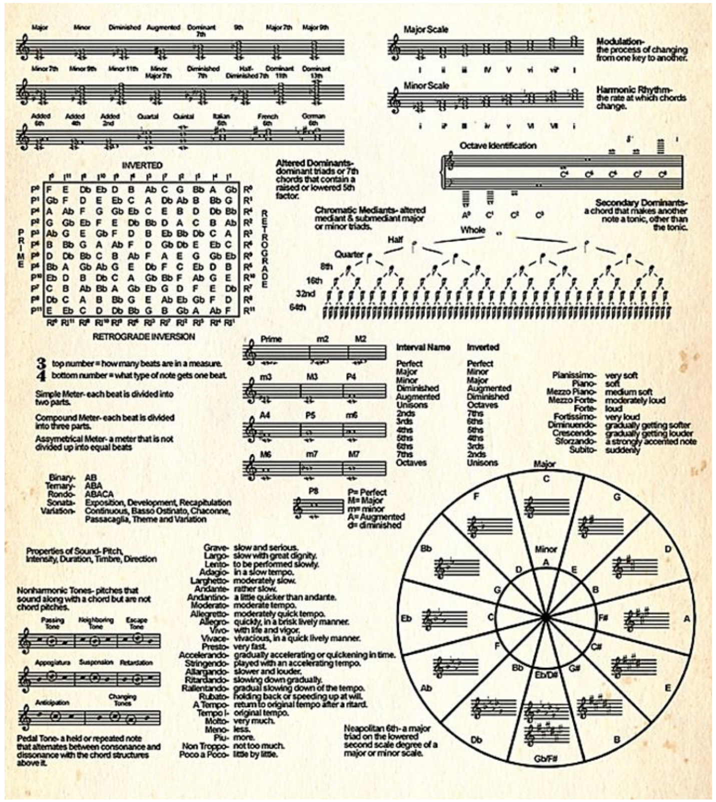
- Need to do self-study to cover the lecture
  - Study data structure by yourself
  - Solve programming problems at [link]
    - Choose 4~6 representative problems for each step.

Topics	Programming	Math	Data Structure
Steps	1, 2, 3, 4, 5, 6, 7, 10	8, 9	17, 18, 21, 23, 27

  - **Must finish this self-study by March!**
  - If you have a plan to feel angry and dissatisfaction without any effort, you’re not ready. **Please drop this course!**

Do Not Have False Expectation

- ❑ It covers the theoretical area of computer science
  - Thus, it doesn't guarantee that you can pass a coding test even though you get Grade A+ in this course



You'll learn about such things...

If you want to do coding well,

Try to solve programming problems as many as possible BY YOURSELF!

In Next Lecture

- **Motivation to algorithm analysis**
  - Why should we analyze algorithms?
    - Especially in terms of efficiency

- **How to analyze algorithms?**
  - Concept of computational complexity
  - Asymptotic analysis

Thank You