



# OT & 시간복잡도

BABO 4기 이론반

분석 22기 원종빈



# CONTENTS

1. OT
2. 공부 방향성 제안
3. 시간복잡도

# 1. OT

---



## OT - 스터디장 소개

- 분석 22기 원종빈
  - 고려대학교 언어학/인공지능 전공
  - 서울대학교 데이터사이언스대학원 SKI-ML Lab 인턴 중
  - 관심분야: LLM, RAG 등 NLP 전반
  - 코테 경험 1회 (LG전자), 자구알 수강, 바보 3기, 백준은 매일 풀려고 노력중!
- 첫 날인만큼, 다른 분들도 간단히 소개해주시면 좋을 것 같습니다~!



## OT - 진행방식

1. 활동은 매주 일요일 늦은 9시~10시, 비대면
2. 매주 발제자를 선정/초청해 다음의 이론들에 대해서 **30분 내외로 발제**
  - 발제 자료는 이론반-발제자료 폴더에 정리해주세요
3. 발제자 외 팀원들은 발제자가 준비한 자료(혹은 이코테 교재)를 미리 **훑어보고**, 발제자가 선정한 **연습문제를 풀어서 Github에 push해야함**.
  - 각자 풀 문제는 이론반-원종빈처럼 개인 폴더에 정리해주세요. (ipynb 권장)
4. 발제자가 발제한 문제들은 다음 시간에 팀원 중 랜덤으로 뽑아서 풀이 및 생각 공유



## OT - 발제자 선정

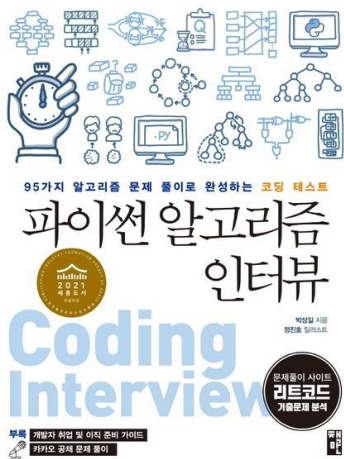
주차	일시	주제	발제자	비고
1주차	7/28	OT, 시간복잡도	원종빈	
2주차	8/4	그리디		
3주차	8/11	DFS/BFS	원종빈	
4주차	8/18	이진탐색		
5주차	8/25	다이나믹 프로그래밍	유종문 (예정)	
6주차	9/1	최단경로 알고리즘	이상윤 (예정)	
7주차	9/8	TBA		

## 2. 공부방향성 제안

---

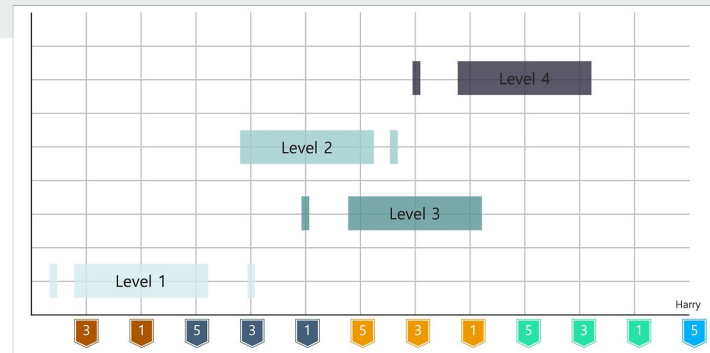
## 공부방향성 제안

- <이것이 취업을 위한 코딩테스트다> (이하 이코테) → 알고리즘 이론을 입문하기에 좋다!





## 공부방향성 제안



- 문제풀이는 백준 + solved.ac 조합을 추천한다!
  - solved.ac 클래스부터 하나씩 미는 게 좋은 것 같다!
  - 이때, 매번 제출-실패를 반복 X, 스스로 테스트케이스 & 디버깅 O
- 참고: 릿코드, 프로그래머스
  - 보통 실제 코테에서는 프로그래머스 플랫폼을 자주 활용한다!
  - 보통의 코테는 프로그래머스 lv 2~3 수준으로 출제된다고 한다!



## 공부방향성 제안

- 내가 맞게 풀었는지, 좋은 코드인지는 어떻게 알 수 있을까?
- 결국 꾸준함이 제일 중요한 것 같다!
  - 문풀반에 놀러가기 (매주 토요일 15~16시)
  - 각자 BABO Github에 PR/Push 해보기
    - <https://github.com/BOAZ-bigdata/24-2 Study BABO Algorithm>

### 3. 시간복잡도

---

# 시간 복잡도(Time Complexity)란?

- 시간 복잡도: 해당 알고리즘을 수행하는 데 필요한 연산의 횟수/소요 시간
  - (CPython 3.7 기준) 초당 20\_000\_000(2천만)번의 연산을 수행한다.
- 공간 복잡도: 해당 알고리즘을 수행하는 데 필요한 메모리의 양
- 백준 10818번: 최소, 최대

최소, 최대 성공

 브론즈 III

시간 제한	메모리 제한	제출
1 초	256 MB	398315

입력

첫째 줄에 정수의 개수  $N$  ( $1 \leq N \leq 1,000,000$ )이 주어진다. 둘째 줄에는  $N$ 개의 정수를 공백으로 구분해서 주어진다. 모든 정수는  $-1,000,000$ 보다 크거나 같고,  $1,000,000$ 보다 작거나 같은 정수이다.

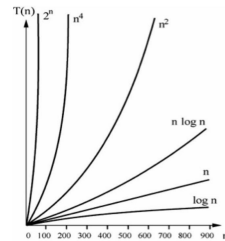
# Big-O 표기법

- 어떠한 알고리즘이 가지는 시간복잡도의 **상한선(upper-bound)**, 즉 최악의 경우!
- 보통은 가장 차수가 높은 항만 고려한다! (계수 역시 무시하는 경우도 많음)
  - $O(n^2)$ ,  $O(n)$ ,  $O(2^n)$ ...

```
n = 10_000_000
for i in range(n):
    i += 10
    i /= 2
    print(i)
```

## Typical Efficiency Classes

➤ Constant	$O(1)$
➤ Logarithmic	$O(\log(n))$
➤ Linear	$O(n)$
➤ Polylogarithmic	$O(\log^k(n))$
➤ Quadratic	$O(n^2)$
➤ Cubic	$O(n^3)$
➤ Polynomial	$O(n^k)$ for any $k > 0$
➤ Exponential	$O(2^n)$ , $O(k^n)$ $k > 1$
➤ Super-Exponential	$O(2^{f(n)})$ for $f(n) = n^{(1+\epsilon)}$ , $\epsilon > 0$





## Big-O 표기법

- N의 범위가 500인 경우: 시간 복잡도가  $O(N^3)$ 인 알고리즘을 설계하면 문제를 풀 수 있다.
- N의 범위가 2,000인 경우: 시간 복잡도가  $O(N^2)$ 인 알고리즘을 설계하면 문제를 풀 수 있다.
- N의 범위가 100,000인 경우: 시간 복잡도가  $O(N \log N)$ 인 알고리즘을 설계하면 문제를 풀 수 있다.
- N의 범위가 10,000,000인 경우: 시간 복잡도가  $O(N)$ 인 알고리즘을 설계하면 문제를 풀 수 있다.

```
# 매번 입력된 숫자를 리스트에 추가해, 그 합계를 구하는
알고리즘
n_inputs = int(input())
nums = []
for _ in range(n_inputs):
    nums.append(int(input()))
print(sum(nums))
```



## Brute-force(완전탐색) 알고리즘

- 모든 경우의 수를 brutally(무지성으로) 다 세보는 알고리즘
- 코드 짜기는 쉽지만, 정작 해당 조건을 완전탐색으로 풀어야하는지를 판단하기 쉽지 않다!
  - 시간복잡도를 활용해 추정해보자!
- 백준 1018번: 체스판 다시 칠하기
- small tip: `import sys; input = lambda: sys.stdin.readline().rstrip()` makes input faster! (only BOJ)
  - 출처: <https://dim03178.tistory.com/21>



# Time Complexity is not easy!

- 알고리즘을 공부한다 == 주어진 시간복잡도 내에 문제를 해결한다
  - 시간복잡도의 제한만 없다면, 사실 모든 문제를 brute-force로 해결하는 것도 가능하다!
- 그렇기에 알고리즘/코드를 공부할 때, 해당 코드의 시간복잡도도 함께 공부 하자!
- 그리고 정작 문제를 풀었더라도, 시간복잡도를 더 줄일 수 있는 테크닉/방법은 없는지도 고민하자!
- 첨부한 `1_time_complexity.md` 파일이 도움이 되었으면 좋겠습니다!



# 과제 공지

1. 백준 1018번: 체스판 다시 칠하기 (S4)
2. 백준 18870번: 좌표 압축 (S2)
3. 프로그래머스 - 연속된 부분 수열의 합 (lv2)