## 파이썬 코딩 심화반 1주차

1. 기업의 **코딩 테스트 문제를 바탕**으로 기존에 배웠던 내용에 더하여 **알고리즘과 자료구조 공부까지** 하고자 함.

2. 삼성전자나 카카오, 네이버와 같은 회사에 개발자 또는 언어공학으로 입사하기 위해서는 코딩 테스트가 필수! 문제 해결 능력을 평가한다.

3. 모임은 일주일에 한 번(목요일 오후 2시), 일단은 비대면으로 하고, 코로나 상황을 지켜보면서 대면으로 할 수도 있음.

4. 스터디 진행은 '기초 다지기', '심화 다지기' 두 파트로 나뉜다.

   - 온라인: **리플릿**(온라인 개발환경), **코드업/백준**(온라인 저지 시스템) ->기초 다지기
   - 책: 나동빈의 '**이것이 코딩 테스트다**' -> 심화 다지기

5. Schedule

**[기초 다지기]** 파이썬 문법 공부 + **백준**에서 쉬운 문제부터 50문제 가량 풀기

**[심화 다지기]** 유형별 알고리즘(2부)와 기출문제(3부) 학습 → 백준에서 유형별 문제 5개 이상 풀기

**[책 완독 후]** 백준 온라인 저지에서 삼성 SW 역량 테스트 문제집 풀기
→ 프로그래머스에서 카카오 문제 풀기
→ 책의 2부와 3부를 중심으로 주요 알고리즘 유형 복습하기]

- 주차 별 학습내용

| 주차 | 내용 |
|---|---|
| 1주차(1/21) | Introduction |
| 2주차(1/28) | 백준(입출력 사칙연산, 1차원 배열, 문자열, if, for, while)<br>교수님 교안(문자열(1), (2), 리스트 튜플 집합 딕셔너리, 흐름제어) |
| 3주차(2/4) | 2주차 어려운 것 복습 + 백준(함수) + 교수님 교안(함수) |

➔ 기초 다지기: 각자 백준 문제를 풀어보고 코드공유하고, 어려웠던 점이 있으면 논의해 본다.

| 주차 | 내용 |
|---|---|
| 4주차(2/11) | Ch. 03 그리디 [이론+문제] —설날?? |
| 5주차(2/18) | (그리디 기출문제) + Ch. 04 구현 [이론+문제] |
| 6주차(2/25) | (구현 기출문제) + Ch. 05 DFS/BFS [이론+문제] |
| 7주차(3/4) | (DFS/BFS 기출문제) + Ch. 06 정렬 [이론+문제] |
| 8주차(3/11) | Ch. 07 이진탐색 [이론+문제] |
| 9주차(3/18) | Ch. 08 다이나믹 프로그래밍 [이론+문제] |
| 10주차(3/25) | Ch. 09 최단경로 [이론+문제] |
| 11주차(4/1) | Ch. 10 그래프 이론 [이론+문제] |

➔ **미정(기초 이후에 방식과 스케줄은 다시 논의)**

➔ 한 챕터당 한 주씩 나가려고 생각중이나 스케줄은 바뀔 수도 있다! **한 챕터를 각자 공부해 온 후(유튭 영상보기, 독학, 문제풀기), 한 주에 한 사람씩 한 챕터를 돌아가면서 몇 개 문제를 설명하고, 서로 코드 공유하기**

6. **★복습하기★**

   - 책 문제를 푼 다음 온라인 저지 사이트에서 동일 유형의 문제를 풀기(이론 학습 → 기출 학습 → 온라인 저지 사이트)
   - 개인 블로그나 깃허브에 기록 남기기. 자신만의 방법으로 자신이 푼 문제나 이해한 알고리즘 내용을 기록하기. 코딩 테스트 직전에 훌륭한 요약집이 될 것.
   -

7. Sign-up

   개발환경

   - Repl.it 회원가입: https://repl.it/

   온라인 저지 시스템

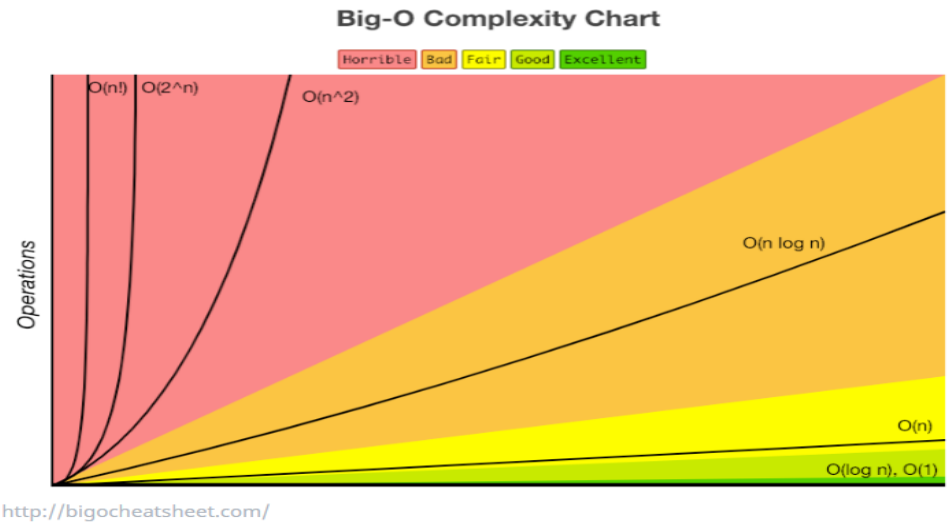
   - 코드업 회원가입: https://codeup.kr/index.php
   - 백준 회원가입: https://www.acmicpc.net/

8. Part.1 Summary

- 코딩테스트란? 기업/기관에서 직원을 채용하기 위해 시행되는 문제 풀이 시험. 채용 과정에서 지원자의 문제 해결 능력을 알아볼 수 있고, 응시자 수를 효과적으로 줄일 수 있다는 장점이 있다.
    - 채점 시스템: **온라인**(인터넷 검사 허용-> 자신만의 코드로 표현하는 능력이 중요) / **오프라인**(검색 허용 x, -> 낯선 시험 환경, 알고리즘 해결 과정 설명)
    - 온라인 저지: 프로그래밍 대회나 코딩 테스트에서 나올 법한 문제를 시험해보는 온라인 시스템 (백준, **코드업(코딩테스트 초보자)**, 프로그래머스)
    - 온라인 개발 환경: 리플릿(추천, https://repl.it/languages/python3?v2=1), 파이썬 튜터(실행 과정을 상세히 알려줌, 파이썬 초보 추천), 파이참(오프라인, 디버깅),
    - **자신만의 소스코드 관리하기 / 팀 노트**: 자주 사용하는 코드를 라이브러리화 하기, 코딩 테스트를 준비하다 보면 알고리즘이 같은 패턴으로 사용되는데, 이 때 더 빠르게 코드를 구현할 수 있다는 장점이 있다.

      (https://github.com/ndb796/Python-Competitive-Programming-Team-Notes)
    - 기출 문제 유형? 그리디, 구현, 탐색, 정렬 등. 회사 마다 상이

- 복잡도
    - 복잡도(Complxity): 시간 복잡도(수행시간) / 공간 복잡도(메모리 사용량). 복잡도가 낮을 수록 좋은 알고리즘. '소스코드가 복잡해 보인다'와 다른 의미. 코딩테스트에서 복잡도란 별 말이 없다면 시간 복잡도를 의미한다.
    - **빅오 표기법(Big-O Notation)**: 가장 빠르게 증가하는 항만을 고려하는 표기법. 함수의 상한만을 나타낸다. 함수에서 가장 큰 차수만 남긴다. 계수도 제거. (극한의 개념) N이 가장 큰 수라고 가정하면, 상한만을 가정한다고 하더라도 복잡도를 계산할 수 있다. Ex) n^3 + 3
    - 복잡도 표기




-

**상수시간**: O(1)-몇 번의 연산만 거치면 수행이 완료되는 경우 → **로그시간**: logN에 비례하는 만큼 수행이 완료된다. 문제를 해결하기 위해 필요한 단계들이 연산마다 특정 요인에 의해 줄어듦. → **선형시간**: O(N) – O(N) – linear time (N=데이터 크기). 문제를 해결하기 위한 단계의 수와 입력값 n이 1:1의 관계를 가진다.

EX) N개의 데이터의 합을 계산하는 프로그램 예제.

Array = [3, 5, 1, 2, 4] (N = 5)

```
Sum = 0

for x in array:

    Sum += x
```

print(summary) -→ array에 있는 데이터를 하나씩 확인하여 가져옴. 따라서 이 코드의 수행시간은 데이터의 개수 N에 비례할 것임을 알 수 있다(리스트의 원소 개수에 비례한다.).

EX 2) 이중 반복문을 이용하는 프로그램.

```
for i in array:

    for j in array:

        temp = i * j
```

print(temp) → i 하나씩 순회할 때마다, j가 또한 한번씩 순회한다. 따라서 5*5. O(N^2). 모든 이중 반복문의 시간복잡도가 O(N^2)인 것은 아니다.

■ 결론

작성한 프로그램이 모든 입력을 받아 이를 처리하고 실행결과를 출력하는 데까지 걸리는 시간을 '시간복잡도'라고 한다. 시간 복잡도는 문제 풀이의 핵심이다. 코딩 테스트에서는 해당 시간 안에 동작하는 프로그램을 작성해야 정답 판정을 받을 수 있다. 문제의 시간복잡도 조건을 보고 얼마나 효율적으로 알고리즘을 작성해야 하는지 알 수 있다.

코딩테스트의 시간제한은 1~5초 가량. 파이썬이 1초에 약 2000만번 연산(5초에 1억번)

연산 횟수가 5억 -> 파이썬에서 5~15초의 시간이 걸린다.

O(N^3), N=5000이라면, 약 1250억 연산횟수, 파이썬에서는 2500초 가량 걸린다.

| N의 max | 빅오 |
|---|---|
| 500 (5백) | $O(N^3)$ |
| 2,000 (2천) | $O(N^2)$ |
| 100,000 (10만) | O(NlogN) |
| 10,000,000 (천만) | O(N) |

➔ 조건을 확인하여 사용할 수 있는 알고리즘을 좁혀나갈 수 있다.

■ 수행시간 측정하기

```
import time
start_time = time.time() #측정시작

list = [1, 2, 3]

for x in list:
    list.pop()
    print(list)

end_time = time.time() #측정종료
print("time:", end_time - start_time)
```

```
[1, 2]
[1]
time: 7.05718994140625e-05
```

- 알고리즘 문제 해결 방법

  **요구사항(복잡도)분석 -> 문제 해결을 위한 아이디어 찾기 -> 소스코드 설계 및 코딩**

  지문 읽기 및 컴퓨터적 사고: 지문을 꼼꼼하게 읽고, 문제를 잘게 분해해보기. 이 과정에서는 무엇이 필요한지 단계별로 생각해보고, 문제 요구사항을 분석해서 어느 정도 성능을 가져야 통과 가능할지 고려하기

  일반적인 문제는 **핵심 아이디어를 캐치**한다면 간결하게 소스코드를 작성할 수 있다. **문제를 먼저 이해하고, 어떤 식으로 코드 작성할 것인지 미리 생각해 놓고 코드를 작성해야 한다. 무작정 코드 작성은 하지 말자**.