**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202211959

이름 : 이 준 하

본인의 Github 주소 : [GitHub - juna2032/data-structure-lab-2025](https://github.com/juna2032/data-structure-lab-2025)

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

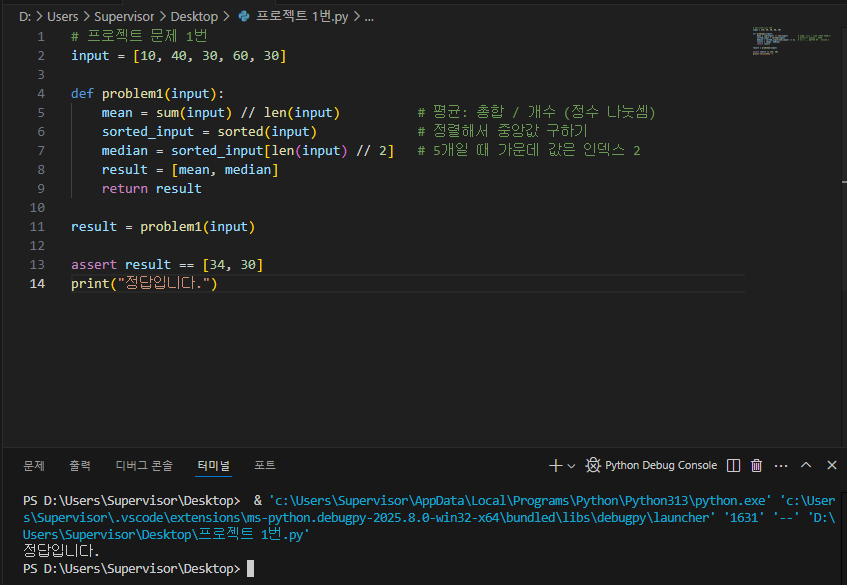
1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

본 문제는 총 5개의 자연수가 주어졌을 때, 이들의 평균과 중앙값을 계산하는 것이 핵심이다. 이를 위해 사용한 주요 데이터 구조는 리스트(list)이다.

* **리스트** 
  + 파이썬의 대표적인 순차 자료구조로, 여러 개의 데이터를 한 번에 저장할 수 있는 자료형이다.
  + 문제에서 주어진 5개의 자연수 [10, 40, 30, 60, 30]을 하나의 리스트로 저장하였다.
  + 리스트는 인덱스를 통해 각 원소에 접근할 수 있으며, sum(), len() 등 다양한 함수와 함께 사용할 수 있어 평균을 구하는 데 매우 효율적이다.
* **정렬된 리스트** 
  + 중앙값을 구하기 위해 기존 리스트를 sorted() 함수를 이용해 오름차순 정렬하였다.
  + 파이썬의 sorted() 함수는 리스트를 정렬한 새로운 리스트를 반환하므로, 기존 리스트는 변경되지 않는다. 이로 인해 원본 데이터를 보존하면서도 정렬된 데이터를 사용할 수 있다는 장점이 있다.
  + 정렬된 리스트에서 중앙값은 리스트의 길이가 홀수일 경우, 중간 인덱스에 해당하는 값을 그대로 꺼내 사용하면 된다.
* **결과 저장용 리스트**
  + 평균과 중앙값 두 값을 함께 반환하기 위해 [0, 0] 형태의 리스트를 초기화하여 결과를 저장하였다.
  + result[0]에는 평균값을, result[1]에는 중앙값을 저장한 뒤 최종적으로 result 리스트를 반환함으로써 두 값을 동시에 처리할 수 있도록 설계하였다.

이와 같이 본 문제는 복잡한 자료구조보다는 **단순하고 효율적인 리스트 구조를 활용하여** 문제를 효과적으로 해결할 수 있도록 구현되었다.

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.



Algorithm Analysis (알고리즘 분석)

**입력**: 자연수 5개 (100보다 작고 10의 배수)

**출력**: 평균값, 중앙값

#### 알고리즘 절차

1. sum() 함수를 이용해 리스트 전체 합을 계산하고, len()으로 나눈 후 정수 나눗셈(//)으로 평균을 구함 → **시간복잡도 O(n)**
2. sorted()를 이용해 입력 리스트를 정렬 → **시간복잡도 O(n log n)**
3. 정렬된 리스트의 가운데 값(인덱스 2)을 선택하여 중앙값 결정 → **시간복잡도 O(1)**
4. 결과를 리스트 형태로 반환 → **시간복잡도 O(1)**

#### 전체 시간복잡도:

* 평균 계산: O(n)
* 정렬: O(n log n)
* 중앙값 선택 및 반환: O(1)  
  → **총합: O(n log n)**

#### 메모리 사용:

* 입력 리스트, 정렬된 리스트, 결과 리스트 등 고정된 길이(5개) 기준의 리스트만 사용되므로 **메모리 사용은 매우 작고 효율적임**

**문제 2.**

1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

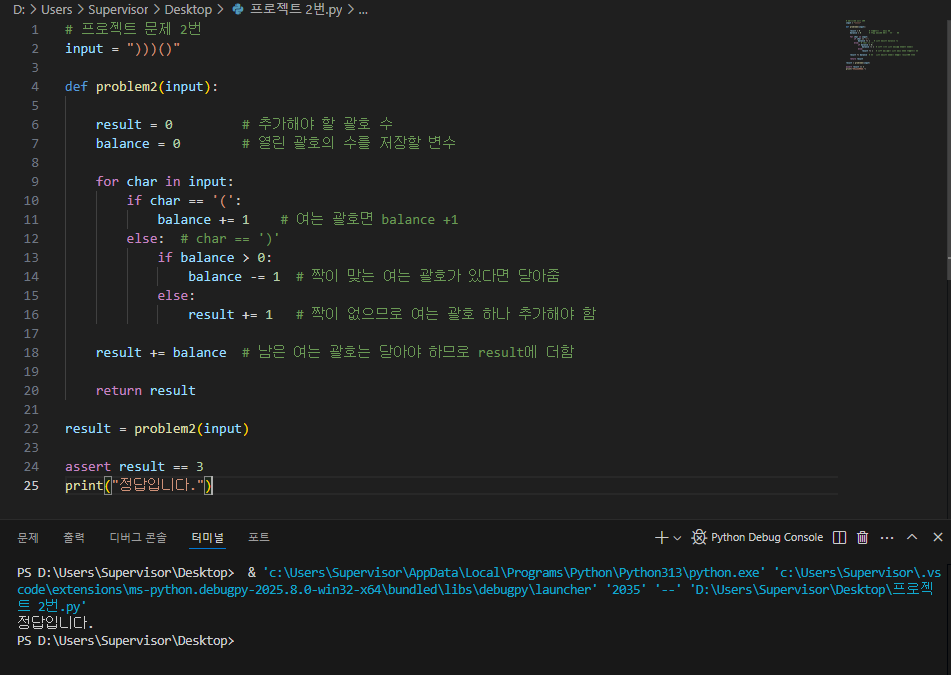
이 문제를 해결하기 위해 사용한 주요 데이터 구조는 **정수형 변수**인 balance와 result이다. 이 두 변수는 괄호열의 상태를 추적하고, 복원에 필요한 괄호 개수를 계산하는 데 핵심적인 역할을 한다.

* **balance**  
  괄호 문자열을 왼쪽에서 오른쪽으로 한 글자씩 순차적으로 탐색하면서, 현재까지 **열린 괄호 '('의 수를 추적**하는 데 사용되는 변수이다.  
  여는 괄호 '('가 등장할 때마다 balance 값을 1씩 증가시키고, 닫는 괄호 ')'가 등장할 때 balance가 1 이상이면 1을 감소시켜 쌍을 맞춘다.  
  이 변수는 괄호의 짝이 맞는지를 실시간으로 판단하는 데 사용되며, 괄호열의 균형 상태를 나타낸다.
* **result**  
  완전한 괄호열을 만들기 위해 **추가로 붙여야 하는 괄호의 총 개수**를 저장하는 변수이다.  
  괄호를 탐색하는 도중, 닫는 괄호 ')'가 나왔을 때 balance가 0이면, 대응되는 여는 괄호가 없다는 뜻이므로 앞에 '(' 하나를 추가해야 하며, 이 경우 result를 1 증가시킨다.  
  또한, 모든 탐색이 끝난 후 balance가 0보다 크다면, 이는 아직 닫히지 않은 여는 괄호가 남아 있다는 의미이므로, 그 수만큼 닫는 괄호를 뒤에 추가해야 하며, result에 balance 값을 더해준다.

이처럼 balance와 result는 각각 **괄호의 균형 상태**와 **추가해야 할 괄호 수**를 추적하며, 문제를 효율적으로 해결하는 데 필요한 최소한의 정보를 저장하는 변수이다.

이 두 변수는 모두 단일 정수 값만을 저장하며, 입력 문자열의 길이와 관계없이 일정한 크기의 메모리만 사용하므로, 해당 알고리즘의 공간 복잡도는 O(1)로 매우 효율적이다.

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.



Algorithm Analysis (알고리즘 분석)

**입력**: 괄호로 이루어진 문자열 1개 (최대 길이 50)  
**출력**: 완전한 괄호열을 만들기 위해 앞과 뒤에 추가로 붙여야 할 괄호의 최소 개수

### 알고리즘 절차:

* 괄호 문자열을 왼쪽부터 오른쪽까지 한 글자씩 탐색하면서 balance 변수를 통해 현재까지 열린 괄호 '('의 수를 추적함 → **시간복잡도 O(n)**
* 닫는 괄호 ')'가 등장했을 때 balance가 0이면, 앞에 여는 괄호가 없다는 의미이므로 result 값을 1 증가시켜 추가가 필요함을 기록함 → **시간복잡도 O(1)**
* 닫는 괄호 ')'가 등장했을 때 balance가 1 이상이면, 기존에 열려 있던 괄호와 쌍을 이루므로 balance를 1 감소시킴 → **시간복잡도 O(1)**
* 모든 문자를 순회한 이후, balance가 0보다 크면 닫히지 않은 여는 괄호가 남아 있는 것이므로 result에 balance 값을 더해 최종 결과를 도출함 → **시간복잡도 O(1)**

### 전체 시간복잡도:

* 괄호 문자열 순회 및 처리: **O(n)**
* 추가 괄호 계산 및 반환: **O(1)**  
  → **총합: O(n)**

### 메모리 사용:

* 입력 문자열 외에는 balance와 result라는 **두 개의 정수형 변수만 사용**하며, 스택이나 리스트 같은 추가 자료구조를 사용하지 않음
* 입력 길이에 상관없이 일정한 공간만 사용하는 구조이므로, 공간복잡도는 O(1)이다
* 따라서 메모리 사용은 매우 작고 효율적인 알고리즘이라고 할 수 있다

**문제 3.**

1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

**차원 리스트 forest**

곰이 탐색하는 공간을 나타내기 위해 사용된 자료구조이다.  
입력으로 주어진 input 리스트를 그대로 복사하여 사용하며, 각 칸에는 0(빈칸), 1~6(벌집 크기), 9(곰의 위치)가 들어 있다.  
곰은 이 리스트를 기준으로 이동하며, 꿀을 먹은 자리는 0으로 변경된다.

**정수형 변수들 (bear\_size, time, honeycomb\_count)**

1.bear\_size: 곰의 현재 크기를 나타내며, 초기값은 2이다.

2.time: 곰이 꿀을 먹기 위해 이동한 전체 시간을 누적 저장한다.

3.honeycomb\_count: 곰이 현재 크기 상태에서 먹은 꿀의 개수이며

이 값이 bear\_size와 같아지면 곰의 크기가 1 증가한다.

**위치 저장 변수 (bear\_x, bear\_y)**  
곰의 현재 위치를 나타내며, 탐색 도중 위치가 이동할 때마다 갱신된다.

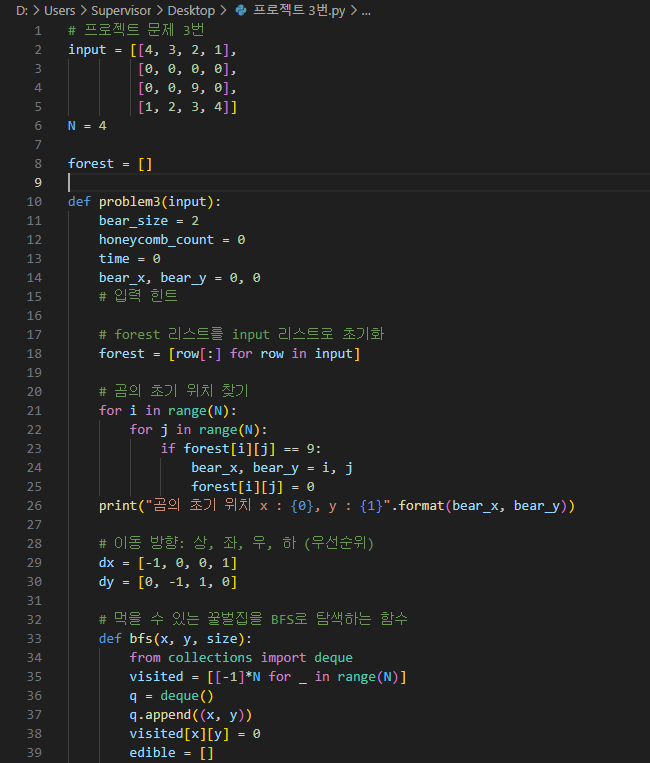
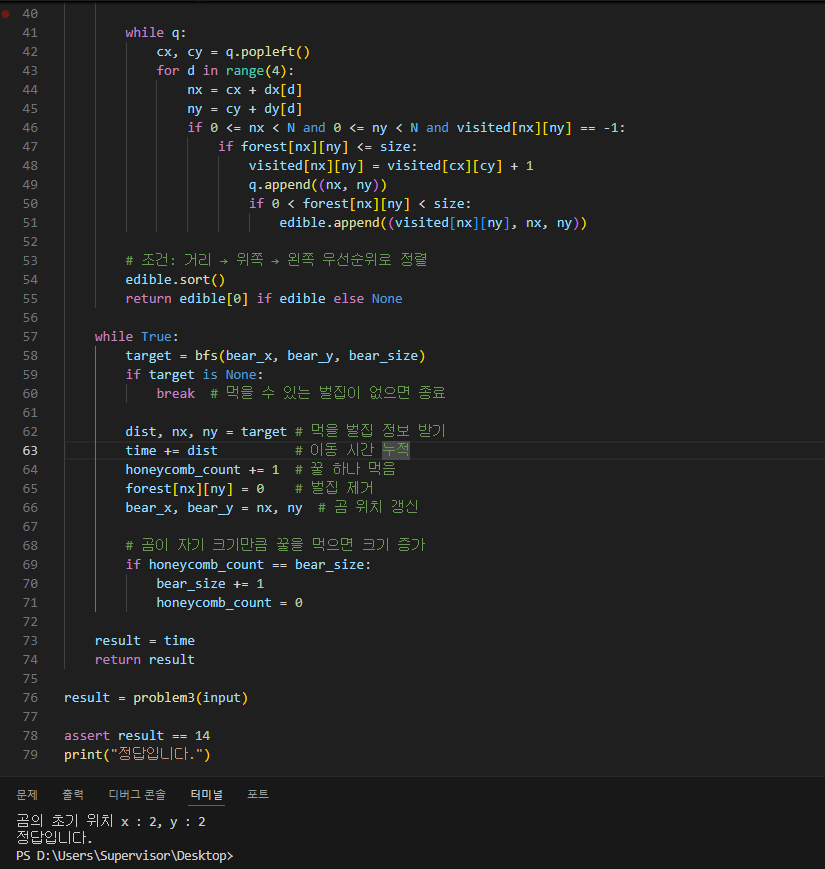
**큐(queue)**:  
BFS(너비 우선 탐색)를 구현하기 위해 collections.deque를 사용하였다.  
이는 탐색 중 가장 가까운 칸부터 순차적으로 방문할 수 있게 해주며, 곰이 가장 가까운 벌집부터 찾아갈 수 있도록 해준다.

**2차원 리스트 visited**  
BFS 탐색에서 각 칸의 방문 여부 및 곰이 도달하는 데 걸린 시간을 기록하기 위해 사용된다.  
초기값은 모두 -1로 설정되며, 방문 시 거리 값을 기록한다.

**리스트 edible**  
곰이 현재 먹을 수 있는 벌집들을 모아두는 리스트이다.  
BFS 탐색 도중 0 < forest[nx][ny] < bear\_size 조건을 만족하는 벌집의 위치와 거리 정보를 이 리스트에 저장한 뒤,  
거리 → 행 → 열 순으로 정렬하여 우선순위가 가장 높은 벌집을 선택한다.

**이처럼 본 문제는 2차원 배열 기반의 시뮬레이션 문제로,  
리스트와 큐 등의 자료구조를 적절히 조합하여 곰의 행동을 정확히 구현하고 있다.**

1. 본인이 작성한 파이썬 Code 캡처 이미지를 첨부하고 Algorithm Analysis를 수행하세요.

## Algorithm Analysis (알고리즘 분석)

**입력**: N×N 크기의 2차원 정사각형 격자 (숲), 값은 0~6 또는 9  
**출력**: 곰이 더 이상 꿀벌집을 먹을 수 없을 때까지 소요된 총 시간 (초)

### 알고리즘 절차:

* 곰의 초기 위치를 찾고, 해당 위치를 (bear\_x, bear\_y)로 저장한 뒤 빈 칸(0)으로 초기화한다.
* 곰의 크기(bear\_size)는 처음에 2이며, 꿀벌집을 먹을 때마다 개수를 세어 크기를 증가시킨다.
* 곰이 먹을 수 있는 꿀벌집을 찾기 위해 **BFS 탐색**을 수행하며,  
  먹을 수 있는 후보들을 모두 edible 리스트에 저장하고 조건에 따라 정렬한다.
* 가장 우선순위 높은 벌집으로 이동하여 꿀을 먹고, 이동 거리만큼 시간을 누적한다.
* 꿀벌집을 먹은 횟수가 곰의 크기와 같아지면 크기를 1 증가시키고, 다시 BFS를 반복한다.
* 더 이상 먹을 수 있는 벌집이 없을 경우 탐색을 종료하고 누적된 시간을 반환한다.

### 전체 시간복잡도:

* **BFS 탐색 1회당** 시간복잡도는 O(N²)이다.
* 전체 시뮬레이션은 벌집의 수(M)만큼 반복되므로, 총 시간복잡도는 O(M × N²)이다.
* 입력 제한이 작기 때문에 (N ≤ 20), 최악의 경우에도 충분히 빠르게 실행된다