**데이터구조 및 프로그래밍실습 3분반**

**설계 프로젝트**

학번 : 202111373

이름 : 오승민

본인의 Github 주소 : OHSEUNGMIN02

(본 과제의 코드를 본인의 Github에 업로드 하세요)

**문제 1.**

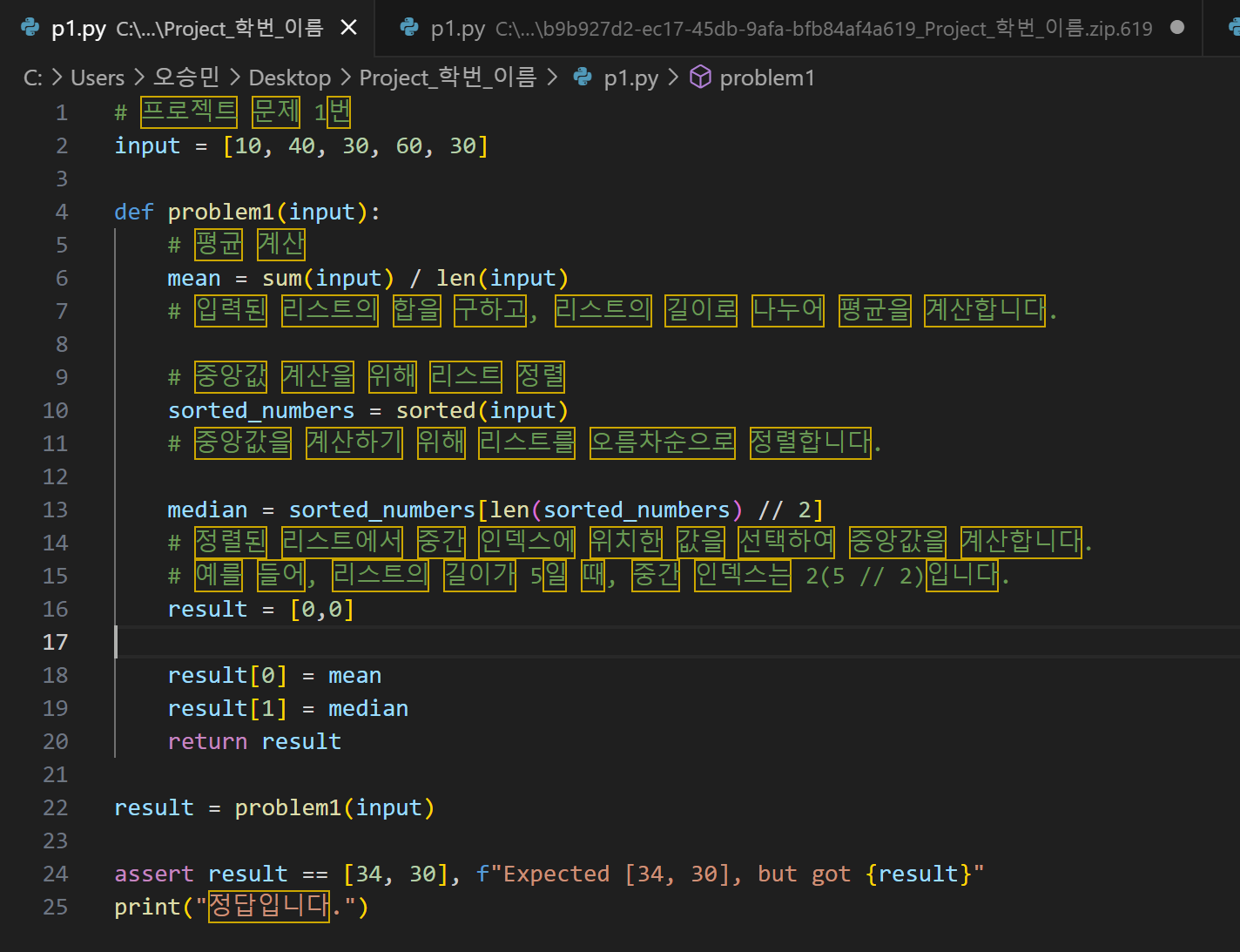
1. 본 문제를 풀기위해 사용한 데이터 구조를 제시하고 간단히 설명하세요.

본 문제를 풀기위해 사용한 데이터 구조는 리스트(List)입니다. 리스트는 여러 값을 순서대로 저장할 수 있는 배열과 유사한 형태를 가지고 있으며 문제 해결 과정에서 사용한 리스트를 설명하면

* input : input = [10, 40, 30, 60, 30]
* 입력 데이터로 사용된 리스트로 여러 숫자 요소를 포함하고 있으며, 평균과 중앙값을 계산하기 위해 사용됩니다.
* sorted\_numbers : sorted\_numbers = sorted(input)
* 중앙값 계산을 위해 정렬된 리스트로 ‘input’ 리스트의 요소를 오름차순으로 정렬한 결과입니다.
* result : result = [0, 0]
* 최종 결과를 저장하기 위한 리스트로 첫 번째 요소는 평균값, 두 번째 요소는 중앙값을 저장합니다.

이와 같이 문제에서도 입력값을 저장하고 , 평균 및 중앙값을 계산하여 결과를 반환하는 데에 리스트를 적절히 활용하였습니다.

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



이 코드의 Algorithm Analysis를 분석해보면 먼저

1. 평균 계산(‘mean’ 변수):

-‘sum(input)’ 함수를 통해 리스트의 모든 요소의 합을 구하며 이 연산의 시간 복잡도는 O(n)입니다. (n은 리스트의 길이

-‘len(input)’ 함수를 통해 리스트의 길이를 구하며, 이는 O(1)의 시간 복잡도를 가집니다.

따라서 평균 계산의 전체 시간 복잡도는 O(n)입니다

1. 중앙값 계산(‘median’ 변수):

-‘sorted(input)’ 함수를 사용하여 입력 리스트를 정렬하며 정렬 알고리즘의 시간 복잡도는 보통 O(n log n)입니다.

-정렬된 리스트에서 중간 위치의 값을 찾는 것은 O(1)의 시간 복잡도를 가집니다.

따라서 중앙값 계산의 전체 시간 복잡도는 O(n log n)입니다.

1. 결과 반환(‘result’ 변수):

-‘result = [mean, median]’을 통해 평균과 중앙값을 저장하고 반환하며 이 연산은 O(1)의 시간 복잡도를 가집니다.

총 시간 복잡도 : 평균 계산과 중앙값 계산을 합하여 총 시간 복잡도는 O(n log n)입니다.

이 알고리즘은 입력의 크기가 커질수록 정렬에 소요되는 시간이 증가하게 되지만, 일반적인 상황에서 빠르게 동작할 수 있는 효율적인 알고리즘입니다.

**문제 2.**

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

본 문제를 풀기위해 사용한 데이터 구조는 스택(stack)입니다. 각 데이터 구조를 설명하면

* 스택(stack)
* stack : 리스트 stack = []로 구현된 스택입니다. 스택은 후입선출(LIFO)원칙을 따르는 데이터 구조로, 여는 괄호’(‘를 만나면 스택에 추가하고, 닫는 괄호’)’를 만나면, ‘close\_needed’를 증가시킵니다.
* 정수 변수
* ‘close\_needed’ : ‘close\_needed = 0’으로 초기화된 변수로, 여는 괄호 없이 등장한 닫는 괄호 ‘)’의 수를 추적하며 이는 추가로 필요한 닫는 괄호의 수를 나타냅니다.
* ‘open\_needed’ : ‘open\_needed = len(stack)’으로 스택에 남아 있는 여는 괄호의 수로, 추가로 필요한 여는 괄호의 수를 나타냅니다.
* ‘result’ : ‘result = open\_needed + close\_needed’로, 최종적으로 추가로 필요한 괄호의 수를 계산하여 반환합니다.

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

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

위 코딩의 Algorithm Analysis를 수행하여 시간 복잡도를 구해보면

1. 반복문

* ‘for char in input’ : 입력 문자열의 각 문자를 순회하며 이 반목문의 시간 복잡도는 입력 문자열의 길이에 비례합니다. 즉, ‘O(n)’입니다. 여기서’n’은 입력 문자열의 길이 입니다.
* 반복문 내에서 각 문자에 대해 일정한 작업을 수행

여는 괄호’(‘인 경우 스택에 추가 : ‘O(1)’

닫는 괄호’)’인 경우 스택이 비어 있는지 확인하고 필요 시 스택에서 제거 : ‘O(1)’

1. 스택 길이 확인

* ‘len(stack)’ : 스택의 길이를 확인하는 연산은 ‘O(1)’입니다.

1. 결과 계산

* ‘open\_needed + close\_needed’: 두 정수의 합을 계산하는 연산은 ‘O(1)’입니다.

전체 시간 복잡도

* 전체 알고리즘의 시간 복잡도는 반복문 내에서 수행되는 작업에 의해 결정되며, 이는 ‘O(n)’입니다.

**문제 3.**

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

본 문제를 풀기위해 사용한 데이터 구조 및 역할을 설명하면

* 리스트
* ‘forest’: 입력된 숲을 복사하여 저장하며 초기 상태에서 곰의 위치(9)를 0으로 변경합니다
* ‘honeycombs’: 곰이 먹을 수 있는 벌집의 위치와 거리를 저장하며 거리를 기준으로 정렬하여 가장 가까운 벌집부터 접근합니다.
* 큐(Queue)
* ‘deque’: BFS를 수행하기위해 사용됩니다.
* 정수 변수
* ‘bear\_size’ : 곰의 현재 크기
* ‘honeycomb\_count’ : 곰이 먹은 벌집의 수
* ‘time’: 총 이동시간
* ‘bear\_x’, ‘bear\_y’ : 곰의 현재위치

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

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

알고리즘 분석으로 시간 복잡도를 구해보면

1. 숲 초기화 및 곰의 초기 위치 찾기:

* 입력된 숲을 ‘forest’로 초기화: ‘O(n^2)’
* 곰의 초기 위치 찾기: ‘O(n^2)’

1. BFS(너비 우선 탐색)

* BFS 탐색: ‘O(n^2)’
* BFS를 통해 각 위치를 한 번씩 방문하며, 방문 가능한 벌집을 찾습니다.

1. 벌집 먹기

* BFS 결과에서 가장 가까운 벌집 찾기 및 이동: ‘O(n^2)’ (최악의 경우)

전체 시간 복잡도:

* 최악의 경우, 모든 위치를 한 번씩 방문하며, BFS를 여러 번 수행해야 할 수 있습니다. 따라서 전체 시간 복잡도는 ‘O(n^4)’ 입니다.