제로 아트 갤러리 문제

문제

제로 아트 갤러리를 운영하고 있는 제로킴은 갤러리에 인터랙티브 아트를 설치하기로 하였다.

이 설치물은 반드시 `2`명이 함께 관람해야 하는 설치물로, `1`명이 먼저 도착하면 다른 한 사람이 도착할 때 까지 대기한다. 하지만 `i`번째로 도착한 사람은 `patience[i]`만큼의 시간이 지나면 기다리다 지루해진 나머지 떠나버린다고 하자.

`i`번째 사람이 관람물에 도착한 시간이 `arrive[i]`로 주어지고, 설치물을 두 사람이 함께 대기 상태가 되면 함께 관람이 가능하다. 세 명이 동시에 설치물에 도달해 있는 경우, `patience[i]`가 더 작은 사람이 우선순위를 가진다.

이 때, 실제로 인터랙티브 아트를 관람한 사람의 수를 구하세요.

단, 아무도 설치물을 관람하지 않았으면 `0`을 반환하고, 정확히 기다림이 끝나는 순간에 다른 사람이 도착하면 관람이 이루어진다.

입력설명

- `0 < patience.length = arrive.length <= 10000`
- `0 <= arrive[i] <= 1000`
- `0 < patience[i] <= 100`

출력설명

관람을 한 인원 수를 정수로 반환

매개변수 형식

- `arrive = [12, 4, 5, 2, 7, 16]`
- `patience = [4, 6, 1, 3, 3, 2]`

반환값 형식

`4`

예제입출력 설명

예제1

- 입력
  - `arrive = [12, 4, 5, 2, 7, 16]`
  - `patience = [4, 6, 1, 3, 3, 2]`
- 반환값
  - `4`
- 설명
  - 다음과 같은 과정을 통해 총 `4`명이 관람하게 된다.

예제2

- 입력
  - `arrive = [9, 3, 8, 5, 7, 0, 10]`
  - `patience = [3, 1, 3, 1, 2, 2, 3]`
- 반환값
  - `4`
- 설명
  - 다음과 같은 과정을 통해 총 `4`명이 관람하게 된다.

제로 카페의 비결 해설

- **Python3 정답 코드 예시**

```
def solution(arrive, patience):
    items = [(a, p) for a, p in zip(arrive, patience)]
    items.sort()

    result = 0

    last_arrive_time, last_patience_time = items[0]
    current_idx = 1
    while current_idx < len(items):
        arrive_time, patience_time = items[current_idx]
        if last_arrive_time + last_patience_time >= arrive_time:
            result += 2
            current_idx += 2
            if current_idx >= len(items):
                break
            last_arrive_time, last_patience_time = items[current_idx -1]
        else:
            current_idx += 1
            last_arrive_time, last_patience_time = arrive_time, patience_time

    return result
```

- **Java 정답 코드 예시**

```
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

class Solution {
    public int solution(int[] arrive, int[] patience) {
        int N = arrive.length;
```

```
    List<Item> items = IntStream.range(0, N)
                    .mapToObj(i -> new Item(arrive[i], patience[i]))
                    .sorted()
                    .collect(Collectors.toList());

    int result = 0;

    int lastArriveTime = items.get(0).arriveTime;
    int lastPatienceTime = items.get(0).patienceTime;
    int currentIdx = 1;

    while (currentIdx < N) {
        int arriveTime = items.get(currentIdx).arriveTime;
        int patienceTime = items.get(currentIdx).patienceTime;
        if (lastArriveTime + lastPatienceTime >= arriveTime) {
            result += 2;
            currentIdx += 2;
            if (currentIdx >= N) break;
            lastArriveTime = items.get(currentIdx - 1).arriveTime;
            lastPatienceTime = items.get(currentIdx - 1).patienceTime;
        } else {
            currentIdx += 1;
            lastArriveTime = arriveTime;
            lastPatienceTime = patienceTime;
        }
    }

    return result;

}

class Item implements Comparable<Item> {
    int arriveTime;
    int patienceTime;

    public Item(int arriveTime, int patienceTime) {
        this.arriveTime = arriveTime;
        this.patienceTime = patienceTime;
    }

    @Override
    public int compareTo(Item o) {
```

```
            int comp = Integer.compare(this.arriveTime, o.arriveTime);
            if (comp == 0) {
                comp = Integer.compare(this.patienceTime, o.patienceTime);
            }
            return comp;
        }

    }
}
```