

알고리즘2 (2024-2)

## 2. 순차 자료구조의 활용

국립금오공과대학교 컴퓨터공학과

김 경 수

# 학습 목표

- ① 순차 자료구조에서 자주 사용되는 투 포인터와 슬라이딩 윈도우의 개념을 이해하고 활용 방법을 익힐 수 있다.
- ② 스택과 큐를 이용하여 문제를 해결하는 방법을 이해하고 실제 문제에 적용할 수 있다.
- ③ 순차 자료구조가 적용되는 대표적인 문제들을 살펴보고, 제한된 시간내에 주어진 문제의 답을 구현하여 해결할 수 있다.

# 순차 자료구조 문제의 특성

# 순차 자료구조 문제의 특성

- 순차 자료구조의 경우 입력이 1차원 배열(리스트)로 주어지는 경우가 대부분.
- 이를 Brute-Force method로 해결하는 경우 시간 복잡도가  $O(n^2)$ 으로 나오는 경우가 많으며, 대부분의 경우 이러한 알고리즘은 time-out 제한으로 인하여 통과되지 못함.
  - 이를 방지하기 위해서는 리스트 전체를 훑는 횟수를 1회로 줄여야 한다.
  - 이를 통해, 알고리즘의 시간 복잡도는  $O(n)$  이하로 설계하는 것이 핵심이다.
  - 알고리즘의 시간 복잡도는 문제에 주어진 "입력의 제한 조건"을 통해서 결정할 수 있다.

# 투 포인터와 슬라이딩 윈도우

- 1차원 배열(리스트)를 2회 이상 탐색해야 할 경우,  $O(n)$  시간 복잡도로 탐색을 수행할 때 유용하게 활용되는 도구
- 투 포인터(Two pointers)
  - 순차 리스트의 특정 요소를 가리키는 두 개의 포인터를 활용하여 특정 조건을 만족하는 부분 구간을 효율적으로 탐색하는 경우에 활용
  - 일반적으로 리스트가 **정렬**된 상태이고, 투 포인터에 의해서 결정되는 **구간의 크기(길이)가 가변적(changeable)**인 경우에 활용
- 슬라이딩 윈도우(Sliding window)
  - 투 포인터의 변형으로 기본적인 특성은 투 포인터와 매우 **유사**
  - 단, 투 포인터에 의해서 결정되는 **구간의 크기가 고정적(fixed)**인 경우에 활용

# 투 포인터와 슬라이딩 윈도우

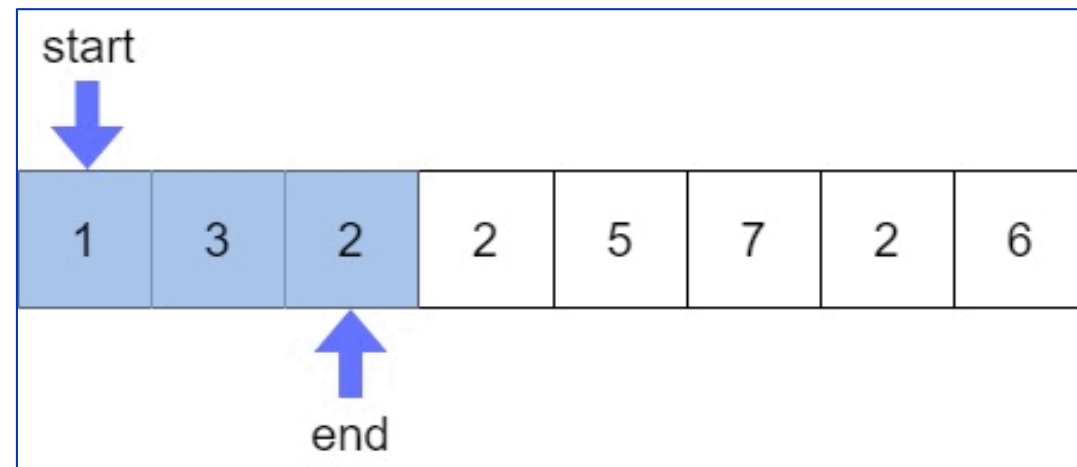
- 투 포인터의 동작 원리



- 슬라이딩 윈도우의 동작 원리

Reference:

[https://adjh54.tistory.com/384#2\)%20%ED%88%AC%20%ED%8F%AC%EC%9D%B8%ED%84%B0%20\(Two%20Pointer%20Algorithm\)%20%EC%A2%85%EB%A5%98-1](https://adjh54.tistory.com/384#2)%20%ED%88%AC%20%ED%8F%AC%EC%9D%B8%ED%84%B0%20(Two%20Pointer%20Algorithm)%20%EC%A2%85%EB%A5%98-1)



# 투 포인터와 슬라이딩 윈도우

- 투 포인터 알고리즘의 일반적인 수행 절차

- ① 리스트의 시작 위치에 첫 번째 포인터와 두 번째 포인터를 설정
- ② 두 포인터 사이의 구간 내 데이터를 조사하고 조건을 확인
  - 조건을 만족할 경우, 원하는 결과를 얻었으므로 알고리즘을 종료
  - 조건을 만족하지 않을 경우, 첫 번째 또는 두 번째 포인터를 이동
- ③ 다시 2번 단계로 돌아가 반복
- ④ 포인터가 리스트의 범위를 벗어나면 알고리즘을 종료

# 스택과 큐

- 스택과 큐는 가장 널리 사용되는 대표적인 선형 자료구조로, 스택, 큐, 원형 큐는 즉석에서 바로 구현할 수 있는 수준에 도달할 때 까지 연습해야 함.
- 스택이 활용되는 문제의 특성
  - 깊이 우선 탐색(DFS), 백트래킹(Backtracking), 재귀 호출 원리가 적용됨
  - 데이터의 흐름을 도식화하면 후입선출(後入先出, LIFO) 구조가 보임
- 큐가 활용되는 문제의 특성
  - 데이터의 흐름을 도식화하면 선입선출(先入先出, FIFO) 구조가 보임
  - 너비 우선 탐색(BFS), 대기열, 순차 처리 등에서 응용
  - 만약, 위와 같은 특성 + 우선순위(priority)가 포함된 문제인 경우 "우선순위 큐(e.g., Heap)"를 활용하는 것이 일반적



# 연습문제 1. 연속된 자연수의 합

# 문제 1. 연속된 자연수의 합

제한시간: 30분

- 어떠한 자연수  $N$ 은 몇 개의 연속된 자연수의 합으로 나타낼 수 있다.
- 당신은 어떤 자연수  $N(1 \leq N \leq 10,000,000)$ 을 몇 개의 연속된 자연수의 합으로 나타내는 가짓수를 알고 싶다. 이때, 사용되는 자연수들은  $N$  이하여야 한다.
- 예를 들어, 15를 나타내는 방법은 15, 7+8, 4+5+6, 1+2+3+4+5의 4가지가 있다. 반면에 10을 나타내는 방법은 10, 1+2+3+4 총 2가지가 있다.
- 자연수  $N$ 을 입력 받으면  $N$ 을 1개 이상(자기 자신인  $N$ 을 포함)의 연속된 자연수의 합으로 나타내는 가짓수를 출력하는 **solution1 함수**를 구현하시오.
- 입력과 출력
  - 입력: 자연수  $N$
  - 출력: 입력된 자연수  $N$ 을 연속된 자연수의 합으로 나타내는 가짓수

입력: $N$	출력
15	4
10	2

# 문제 1. 연속된 자연수의 합 – 문제 분석

## • 문제 분석

### • “연속된 자연수”의 합

➤ 숫자를 일렬로 나열하여 해결할 수 있을 것 같다. ➔ “배열/리스트”를 활용

### • 문제의 제한 조건에서 입력의 길이 $N$ 의 크기는 최대 10,000,000

➤  $O(N)$  이하의 시간 복잡도를 갖도록 알고리즘을 설계해야 함

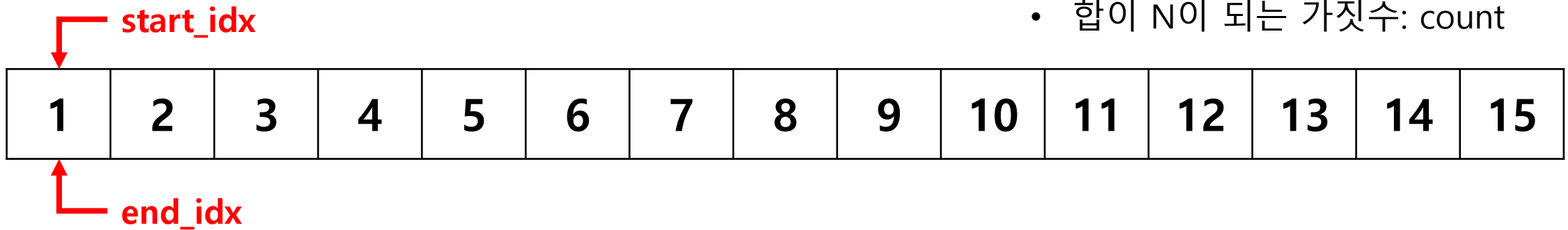
### • 리스트 상에서 연속된 수를 조건에 따라 순서대로 탐색하는 문제이므로, 투 포인터 또는 슬라이딩 윈도우를 활용할 수 있다.

➤ 이때, 구간의 크기가 가변적이므로 투 포인터를 적용하는 것이 바람직하다.

# 문제 1. 연속된 자연수의 합 – 문제 분석

- 손으로 문제 풀어보기 (예제:  $N = 15$ )

- 시작 포인터: `start_idx`
- 마지막 포인터: `end_idx`
- 투 포인터 구간 내 합: `sum`
- 합이  $N$ 이 되는 가짓수: `count`



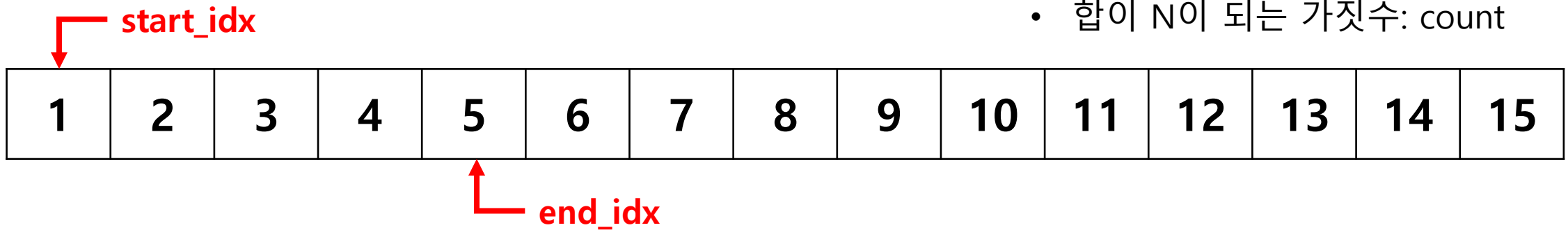
- if `sum_value < N` then  
`sum_value += end_idx; end_idx += 1;`
- if `sum_value == N` then  
`sum_value += end_idx; end_idx += 1; count += 1;`
- if `sum_value > N` then  
`sum_value -= start_idx; start_idx += 1;`

sum_value	1
count	1

# 문제 1. 연속된 자연수의 합 – 문제 분석

## • 손으로 문제 풀어보기 (예제: $N = 15$ )

- 시작 포인터: `start_index`
- 마지막 포인터: `end_index`
- 투 포인터 구간 내 합: `sum_value`
- 합이  $N$ 이 되는 가짓수: `count`



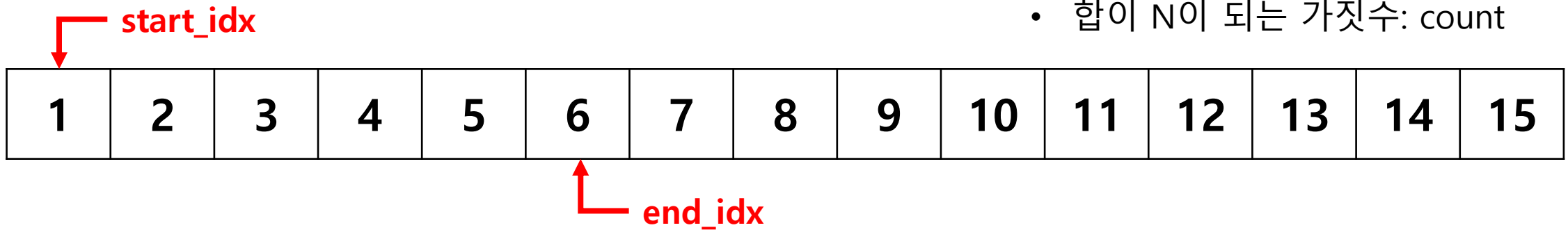
- if `sum_value < N` then  
`sum_value += end_idx; end_idx += 1;`
- if `sum_value == N` then  
`sum_value += end_idx; end_idx += 1; count += 1;`
- if `sum_value > N` then  
`sum_value -= start_idx; start_idx += 1;`

<b>sum_value</b>	<b>15</b>
<b>count</b>	<b>2</b>

# 문제 1. 연속된 자연수의 합 – 문제 분석

## • 손으로 문제 풀어보기 (예제: $N = 15$ )

- 시작 포인터: `start_index`
- 마지막 포인터: `end_index`
- 투 포인터 구간 내 합: `sum_value`
- 합이  $N$ 이 되는 가짓수: `count`



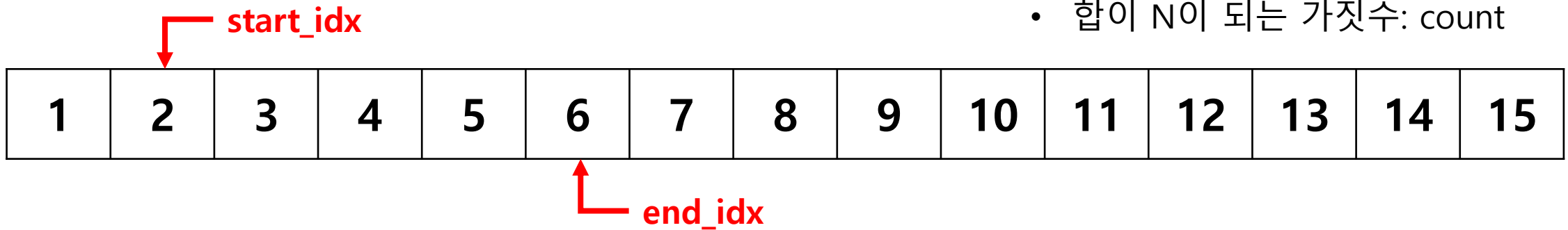
- if `sum_value < N` then  
`sum_value += end_idx; end_idx += 1;`
- if `sum_value == N` then  
`sum_value += end_idx; end_idx += 1; count += 1;`
- if `sum_value > N` then  
`sum_value -= start_idx; start_idx += 1;`

<b>sum_value</b>	<b>21</b>
<b>count</b>	<b>2</b>

# 문제 1. 연속된 자연수의 합 – 문제 분석

- 손으로 문제 풀어보기 (예제:  $N = 15$ )

- 시작 포인터: `start_index`
- 마지막 포인터: `end_index`
- 투 포인터 구간 내 합: `sum_value`
- 합이  $N$ 이 되는 가짓수: `count`



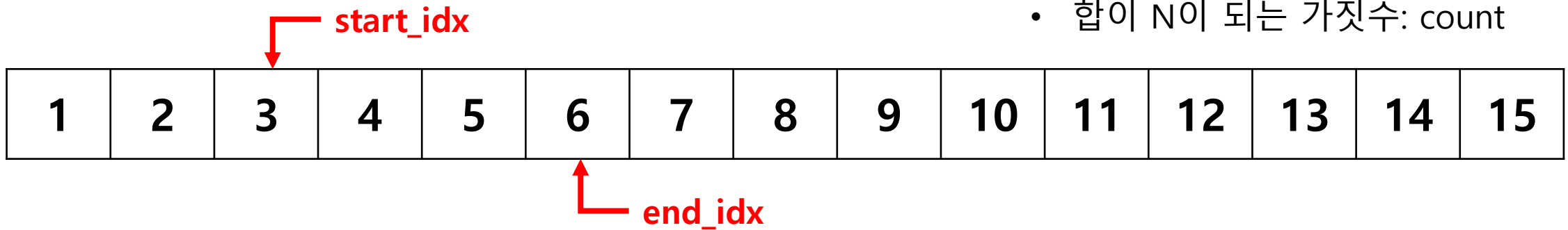
- `if sum_value < N then`  
`sum_value += end_idx; end_idx += 1;`
- `if sum_value == N then`  
`sum_value += end_idx; end_idx += 1; count += 1;`
- `if sum_value > N then`  
`sum_value -= start_idx; start_idx += 1;`

<b>sum_value</b>	<b>20</b>
<b>count</b>	<b>2</b>

# 문제 1. 연속된 자연수의 합 – 문제 분석

## • 손으로 문제 풀어보기 (예제: $N = 15$ )

- 시작 포인터: `start_index`
- 마지막 포인터: `end_index`
- 투 포인터 구간 내 합: `sum_value`
- 합이  $N$ 이 되는 가짓수: `count`



- `if sum_value < N then`  
`sum_value += end_idx; end_idx += 1;`
- `if sum_value == N then`  
`sum_value += end_idx; end_idx += 1; count += 1;`
- `if sum_value > N then`  
`sum_value -= start_idx; start_idx += 1;`

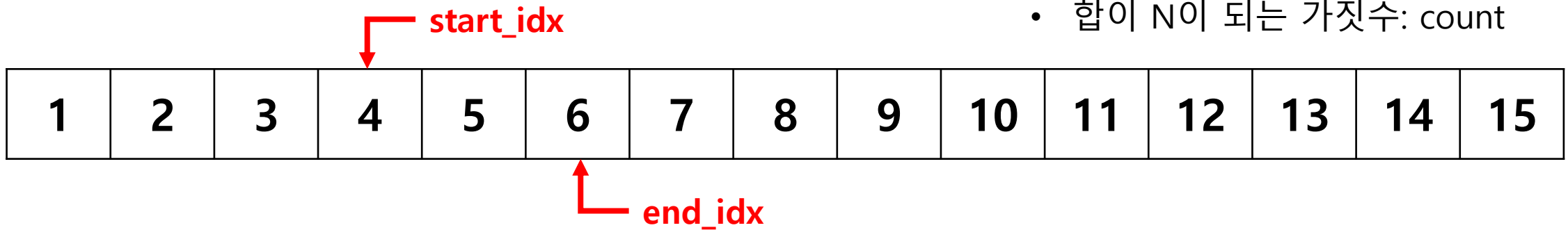
sum_value	18
count	2



# 문제 1. 연속된 자연수의 합 – 문제 분석

## • 손으로 문제 풀어보기 (예제: $N = 15$ )

- 시작 포인터: `start_index`
- 마지막 포인터: `end_index`
- 투 포인터 구간 내 합: `sum_value`
- 합이  $N$ 이 되는 가짓수: `count`



- if `sum_value < N` then  
`end_idx += 1; sum_value += end_idx;`
- if `sum_value == N` then  
`end_idx += 1; sum_value += end_idx; count += 1;`
- if `sum_value > N` then  
`sum_value -= start_idx; start_idx += 1;`

<b>sum_value</b>	<b>15</b>
<b>count</b>	<b>3</b>

# 문제 1. 구현 및 검증

## 테스트 케이스

```
[31] 1 # Test
      2 print( solution1(15) )
      3 print( solution1(10) )
```

```
⇒ 4
   2
```

```
def solution1(N):
    n = int(N)
    count = 1
    start_idx = 1
    end_idx = 1
    sum = 1

    while end_idx != n:
        if sum < n:
            end_idx += 1
            sum += end_idx
        elif sum == n:
            count += 1
            end_idx += 1
            sum += end_idx
        else:
            sum -= start_idx
            start_idx += 1

    return count
```

# 연습문제 2. 모의 고사

# 문제 2. 모의 고사

제한시간: 20분

- 수포자(수학 포기자) 3인은 모의고사의 수학 문제 답안을 전부 찍어서 제출하고자 한다.
- 각 수포자들은 1번 문제부터 마지막 문제까지 다음과 같이 찍는다.
  - 수포자 1의 찍는 방식: 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, ...
  - 수포자 2의 찍는 방식: 2, 1, 2, 3, 2, 4, 2, 5, 2, 1, 2, 3, 2, 4, 2, 5, ...
  - 수포자 3의 찍는 방식: 3, 3, 1, 1, 2, 2, 4, 4, 5, 5, 3, 3, 1, 1, 2, 2, 4, 4, 5, 5
- 1번 문제부터 마지막 문제까지 정답이 순서대로 저장된 리스트 `answers[]`가 주어졌을 때, 가장 많은 문제를 맞힌 사람이 누구인지 반환하는 **solution2 함수**를 작성하시오.
- 제약 조건과 입/출력 예제
  - ✓ 시험은 최대 10,000 문제로 구성된다.
  - ✓ 문제의 정답은 1, 2, 3, 4, 5중 하나이다.
  - ✓ 가장 높은 점수를 받은 사람이 여러 명이라면, 리스트를 오름차순으로 정렬하여 반환하시오.

입력: answers	출력: return
[1, 2, 3, 4, 5]	[1]
[1, 3, 2, 4, 2]	[1, 2, 3]

## 문제 2. 모의 고사 – 문제 분석


### • 문제 분석

- 가장 중요한 포인트: 수포자들의 정답을 찍는 패턴 → 규칙성이 있음.
  - 수포자 1의 찍는 방식: 1, 2, 3, 4, 5 / 1, 2, 3, 4, 5 / ...
  - 수포자 2의 찍는 방식: 2, 1, 2, 3, 2, 4, 2, 5 / 2, 1, 2, 3, 2, 4, 2, 5 / ...
  - 수포자 3의 찍는 방식: 3, 3, 1, 1, 2, 2, 4, 4, 5, 5 / 3, 3, 1, 1, 2, 2, 4, 4, 5, 5 / ...
- 수포자들은 1번부터 위와 같은 패턴을 반복하여 답안을 작성한다.
- 입력으로 주어지는 실제 정답은 1번 부터의 순차적으로 주어진다.
- 중요 사항: 주어진 문제의 개수가 최대 10,000문제이므로 문제의 개수가  $n$ 개일 때,  $O(n)$ 의 시간 복잡도로 알고리즘을 설계해야 한다.

# 문제 2. 모의 고사 – 문제 분석

## • 손으로 문제 풀어보기

- 각 수포자들이 답안을 찍는 패턴과 입력으로 주어지는 실제 정답 리스트 모두 선형 배열이므로, 1번 문항부터 마지막 문항까지 수포자들이 찍은 답안과 실제 정답 리스트를 1대 1로 비교하는 방법으로  $O(n)$  복잡도의 알고리즘을 구현할 수 있다.



수포자 1:	1	2	★3	4	★5	1	2	3	4	★5	1	2
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

수포자 2:	2	1	2	3	2	★4	2	★5	★2	1	2	3
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

수포자 3:	★3	3	1	1	2	2	★4	4	5	★5	★3	3
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

# 문제 2. 모의 고사 – 문제 분석

## • 손으로 문제 풀어보기

- 문제의 제한 조건에 “가장 높은 점수를 받은 사람이 여러 명이라면...”이라는 표현이 있으므로, 점수 계산과 동점자 처리 기능도 구현되어야 함을 알 수 있다.
- 각 문항의 점수는 별도로 주어지지 않았으므로, 모두 1점으로 동일하다고 가정한다.
- 이를 위해 각 수포자들의 취득 점수를 저장하는 변수(배열)을 별도로 마련한다.

수포자 1:	1	2	★3	4	★5	1	2	3	4	★5	1	2
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

수포자 2:	2	1	2	3	2	★4	2	★5	★2	1	2	3
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

수포자 3:	★3	3	1	1	2	2	★4	4	5	★5	★3	3
answers[]:	3	4	3	2	5	4	4	5	2	5	3	1

수포자	1	2	3
점 수	3	3	4

# 문제 2. 구현 및 검증

```
def solution2(answers):
    patterns = [
        [1, 2, 3, 4, 5],
        [2, 1, 2, 3, 2, 4, 2, 5],
        [3, 3, 1, 1, 2, 2, 4, 4, 5, 5] ]
```

```
scores = [0] * 3
```

```
for i, answer in enumerate(answers):
    for j, pattern in enumerate(patterns):
        if answer == pattern[i % len(pattern)]:
            scores[j] += 1
```

```
max_scores = max(scores)
```

```
highest_scores = []
for i, score in enumerate(scores):
    if score == max_scores:
        highest_scores.append(i+1)
```

```
return highest_scores
```

## 테스트 케이스

```
[33] 1 # Test
      2 answers1 = [3, 4, 3, 2, 5, 4, 4, 5, 2, 5, 3, 1]
      3 print( solution2(answers1) )
      4
      5 answers2 = [1, 2, 3, 4, 5]
      6 print( solution2(answers2) )
      7
      8 answers3 = [1, 3, 2, 4, 2]
      9 print( solution2(answers3) )
```

```
⇒ [3]
   [1]
   [1, 2, 3]
```

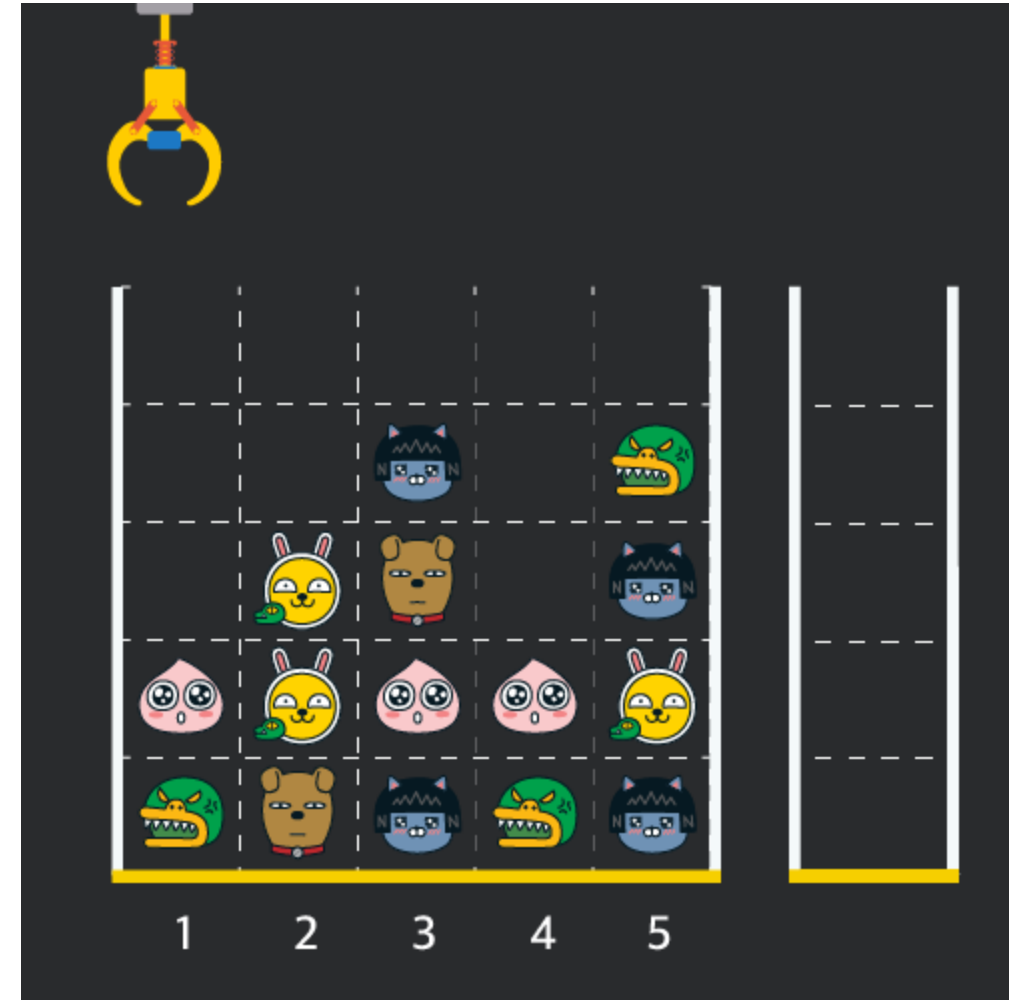


# 연습문제 3. 크레인 인형 뽑기 게임

# 문제 3. 크레인 인형 뽑기 게임 (1/5)

제한시간: 60분

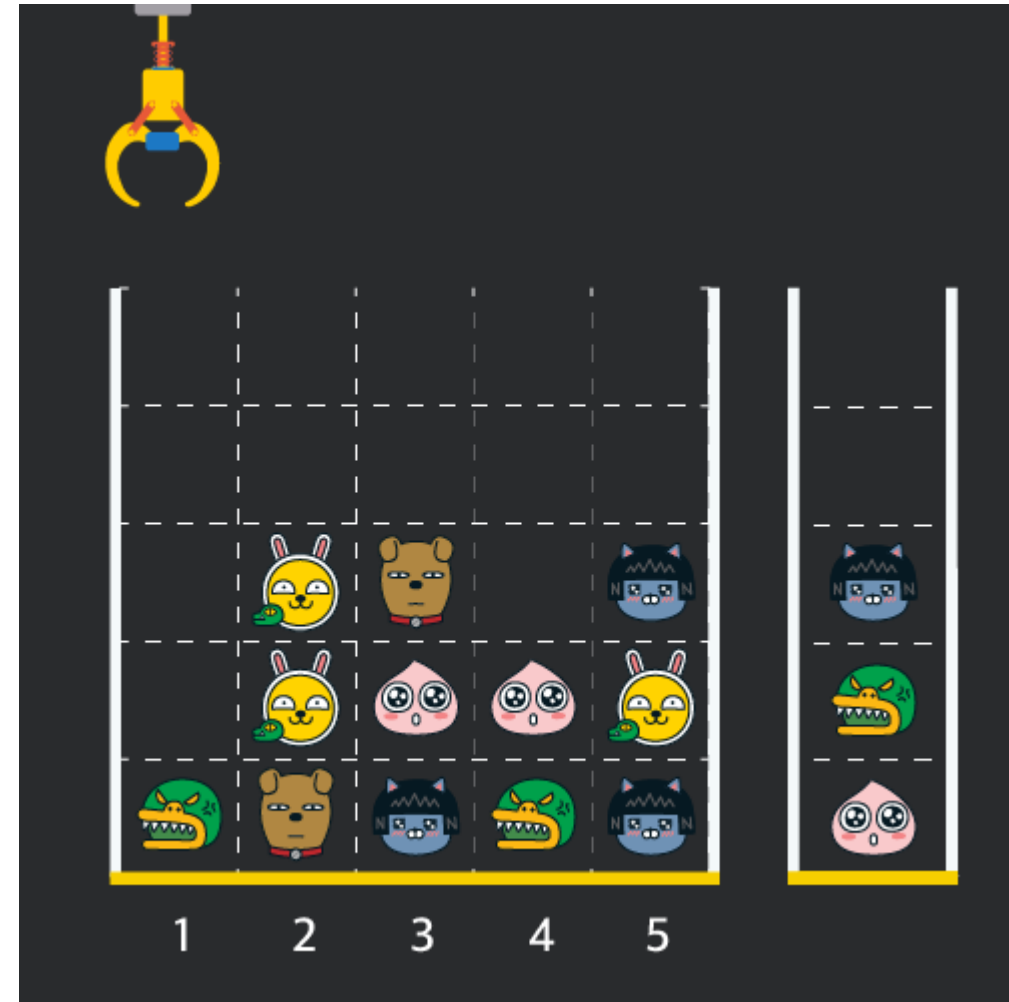
- 게임개발자인 "조르디"는 크레인 인형뽑기 기계를 모바일 게임으로 만들려고 합니다.
- "조르디"는 게임의 재미를 높이기 위해 화면 구성과 규칙을 다음과 같이 게임 로직에 반영하려고 합니다.
- 게임 화면은 "**1 x 1**" 크기의 칸들로 이루어진 "**N x N**" 크기의 정사각 격자이며 위쪽에는 크레인이 있고 오른쪽에는 바구니가 있습니다. (위 그림은 "5 x 5" 크기의 예시입니다).
- 각 격자 칸에는 다양한 인형이 들어 있으며 인형이 없는 칸은 빈칸입니다. 모든 인형은 "1 x 1" 크기의 격자 한 칸을 차지하며 **격자의 가장 아래 칸부터** 차곡차곡 쌓여 있습니다.



# 문제 3. 크레인 인형 뽑기 게임 (2/5)

제한시간: 60분

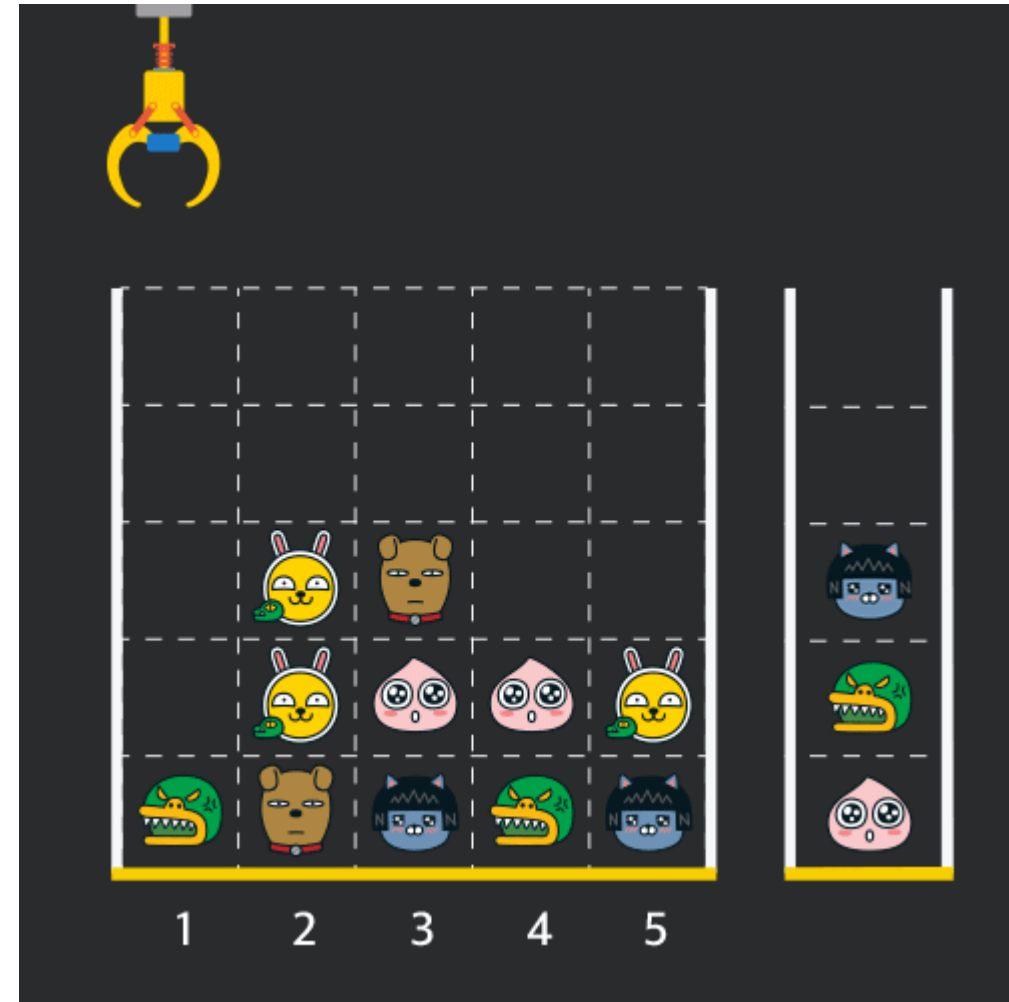
- 게임 사용자는 크레인을 좌우로 움직여서 멈춘 위치에서 가장 위에 있는 인형을 집어 올릴 수 있습니다. 집어 올린 인형은 바구니에 쌓이게 되는 데, 이때 바구니의 가장 아래 칸부터 인형이 순서대로 쌓이게 됩니다.
- 다음 그림은 [1번, 5번, 3번] 위치에서 순서대로 인형을 집어 올려 바구니에 담은 모습입니다.



# 문제 3. 크레인 인형 뽑기 게임 (3/5)

제한시간: 60분

- 만약 같은 모양의 인형 두 개가 바구니에 연속해서 쌓이게 되면 두 인형은 터뜨려지면서 바구니에서 사라지게 됩니다. 위 상태에서 이어서 [5번] 위치에서 인형을 집어 바구니에 쌓으면 같은 모양 인형 **두 개**가 없어집니다.
- 크레인 작동 시 인형이 집어지지 않는 경우는 없으나 만약 인형이 없는 곳에서 크레인을 작동시키는 경우에는 아무런 일도 일어나지 않습니다. 또한 바구니는 모든 인형이 들어갈 수 있을 만큼 충분히 크다고 가정합니다. (그림에서는 화면표시 제약으로 5칸만으로 표현하였음)



## 문제 3. 크레인 인형 뽑기 게임 (4/5)

제한시간: 60분

- 게임 화면의 격자의 상태가 담긴 2차원 배열 board와 인형을 집기 위해 크레인을 작동시킨 위치가 담긴 배열 moves가 매개변수로 주어질 때, 크레인을 모두 작동시킨 후 터트려져 사라진 인형의 개수를 return 하도록 solution3 함수를 완성해주세요.
- 제한사항
  - board 배열은 2차원 배열로 크기는 "5 x 5" 이상 "30 x 30" 이하입니다.
  - board의 각 칸에는 0 이상 100 이하인 정수가 담겨있습니다.
    - 0은 빈 칸을 나타냅니다.
    - 1 ~ 100의 각 숫자는 각기 다른 인형의 모양을 의미하며 같은 숫자는 같은 모양의 인형을 나타냅니다.
  - moves 배열의 크기는 1 이상 1,000 이하입니다.
  - moves 배열 각 원소들의 값은 1 이상이며 board 배열의 가로 크기 이하인 자연수입니다.

- 입출력 예

board	moves	result
[[0,0,0,0,0],[0,0,1,0,3],[0,2,5,0,1],[4,2,4,4,2],[3,5,1,3,1]]	[1,5,3,5,1,2,1,4]	4

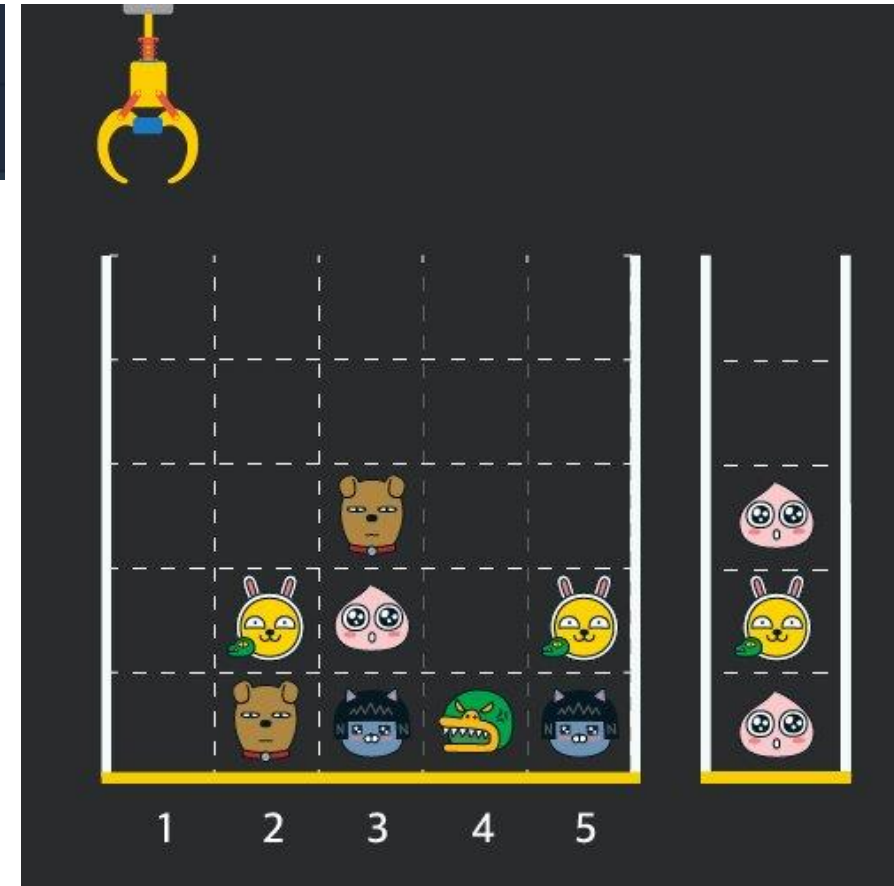
# 문제 3. 크레인 인형 뽑기 게임 (5/5)

제한시간: 60분

board	moves	result
[[0,0,0,0,0],[0,0,1,0,3],[0,2,5,0,1],[4,2,4,4,2],[3,5,1,3,1]]	[1,5,3,5,1,2,1,4]	4

## • 입출력 예제에 대한 설명

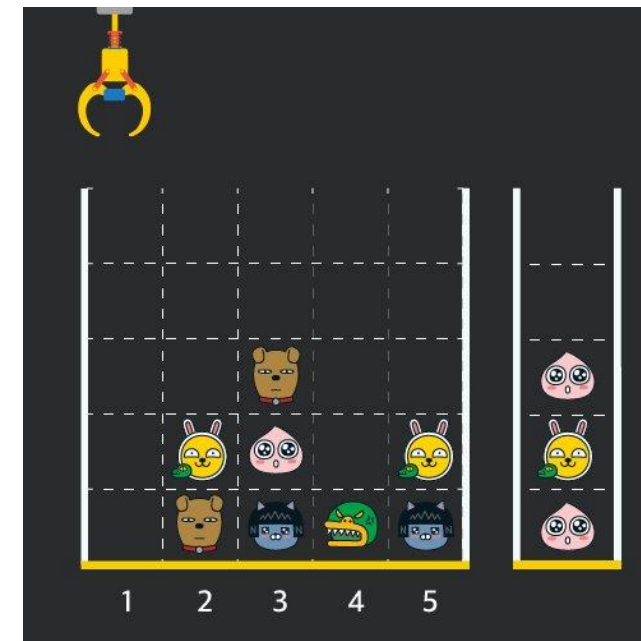
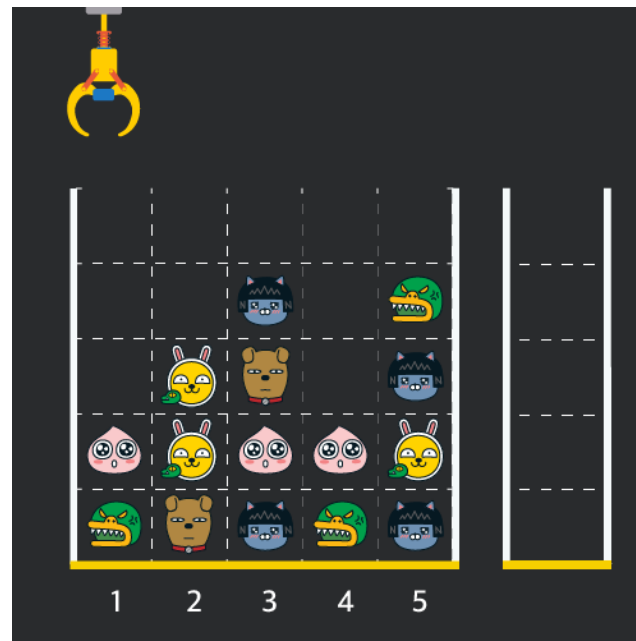
- 인형의 처음 상태는 문제에 주어진 예시와 같습니다.
- 크레인이 [1, 5, 3, 5, 1, 2, 1, 4] 번 위치에서 차례대로 인형을 집어서 바구니에 옮겨 담은 후, 상태는 아래 그림과 같으며 바구니에 담는 과정에서 터트려져 사라진 인형은 4개 입니다.



# 문제 3. 문제 분석

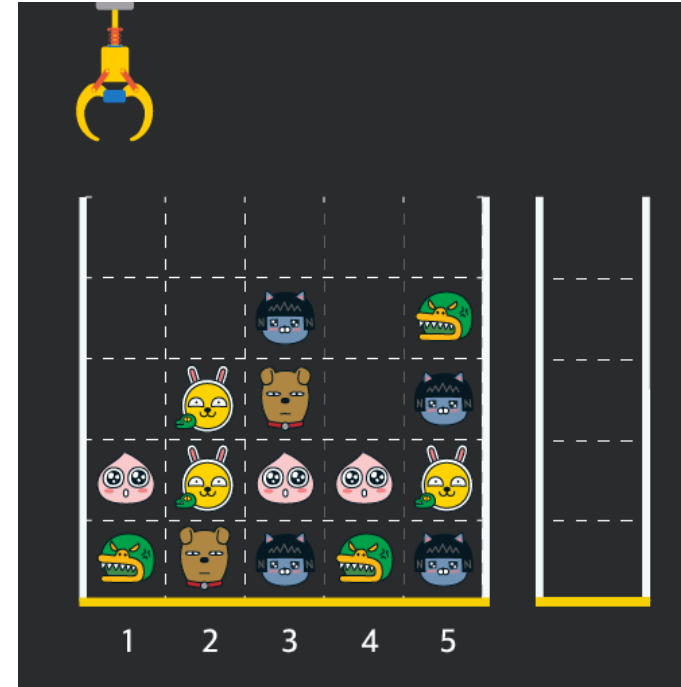
- 문제 지문의 분량이 많으므로 예제를 자세히 분석할 필요가 있다.
- 따라서, 주어진 예제를 하나씩 따라해보면서 어떻게 최종 결과가 도출되는지 우선적으로 확인해본다.

board	moves	result
[[0,0,0,0,0],[0,0,1,0,3],[0,2,5,0,1],[4,2,4,4,2],[3,5,1,3,1]]	[1,5,3,5,1,2,1,4]	4



## 문제 3. 문제 분석

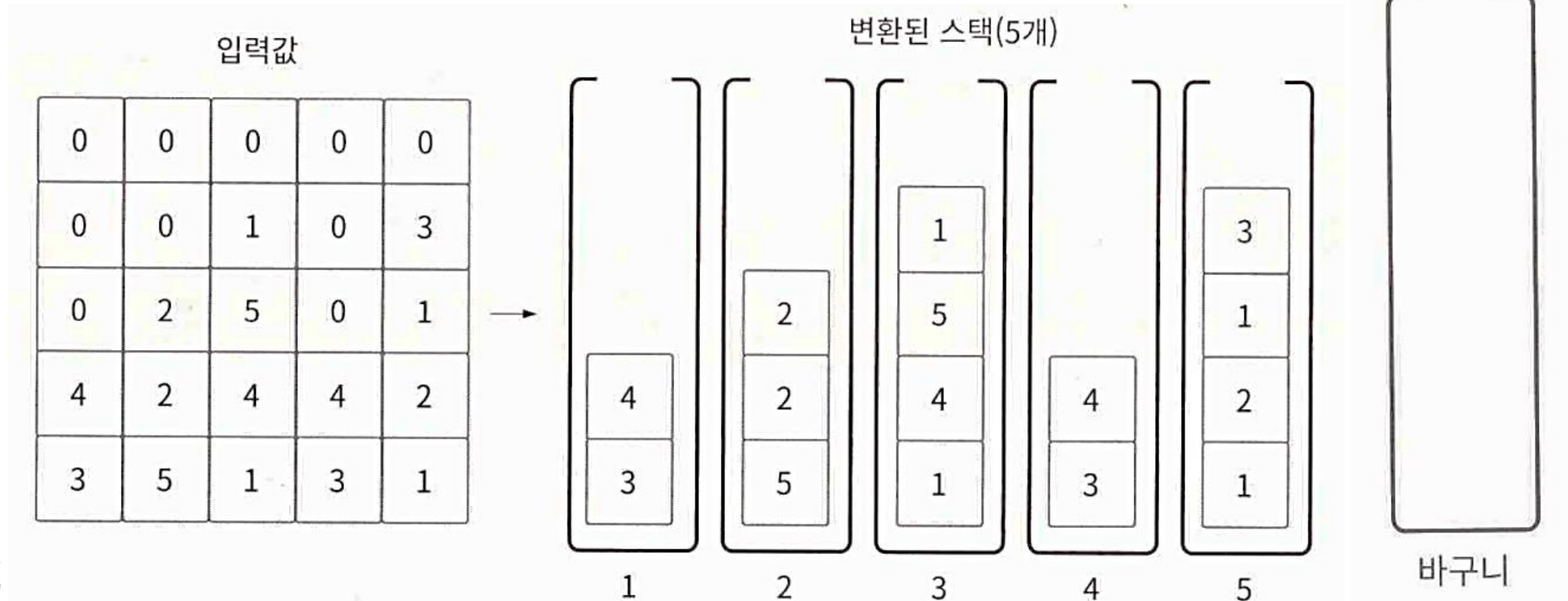
- 문제의 지문에서 설명하는 동작을 그림과 함께 분석한다.
  - 문제를 자세히 읽어보면 **핵심 키워드**가 존재한다.
    - ✓ "격자의 가장 아래 칸부터 차곡차곡 쌓여 있고, ..."
    - ✓ "바구니의 가장 아래 칸부터 인형이 순서대로 쌓인다."
    - ✓ 예시로 주어진 그림들...
  - 예시로 주어진 인형들과 바구니를 **자료구조**로 재구성한다.
    - ✓ 인형이 들어있는 board의 각 열(column)은 스택으로 변환하면 더욱 효율적이다.
    - ✓ Board 내 인형들은 1 ~ 4의 자연수로 표기한다.
    - ✓ Board에서 꺼낸 인형을 담는 바구니인 bucket은 아래에서부터 위로 쌓이고, 오직 맨 위에서만 추가/삭제가 가능하므로 전형적인 스택 구조이다.





## 문제 3. 문제 분석

- 입력 값으로 주어지는 board는 이차원 배열로 바로 사용해도 가능하지만, **각 열(column)을 스택으로 변환**하면 보다 쉽게 구현할 수 있다.
- 바구니인 bucket 또한 **스택으로 구현**한다.



## 문제 3. 문제 해결 전략 수립

- 주어진 문제에서 요구하는 핵심 동작을 자료구조로 표현해보자.

### ① 로봇 팔을 움직여서 board의 m번째 열에서 인형을 꺼내는 동작

- m번째 열에 해당하는 스택에서 꺼낸 값을 doll로 지정한다.

**doll = cols[m].pop()**

### ② 꺼낸 인형과 바구니 최상단의 인형을 비교하는 동작

- Board에서 선택한 값 doll과 바구니에 해당하는 스택 bucket의 최상단에 위치한 값(top)을 비교한다.

**if doll != bucket.top() then ... else ...**

## 문제 3. 문제 해결 전략 수립

### ③ Board에서 꺼낸 인형을 바꾸니 bucket에 담는 동작

- 두 값이 서로 다를 경우 board에서 꺼낸 doll을 bucket에 push한다.

**if doll != bucket.top() then bucket.push(v);**

### ④ 상자에서 꺼낸 인형과 바꾸니의 인형이 터지면서 사라지는 동작

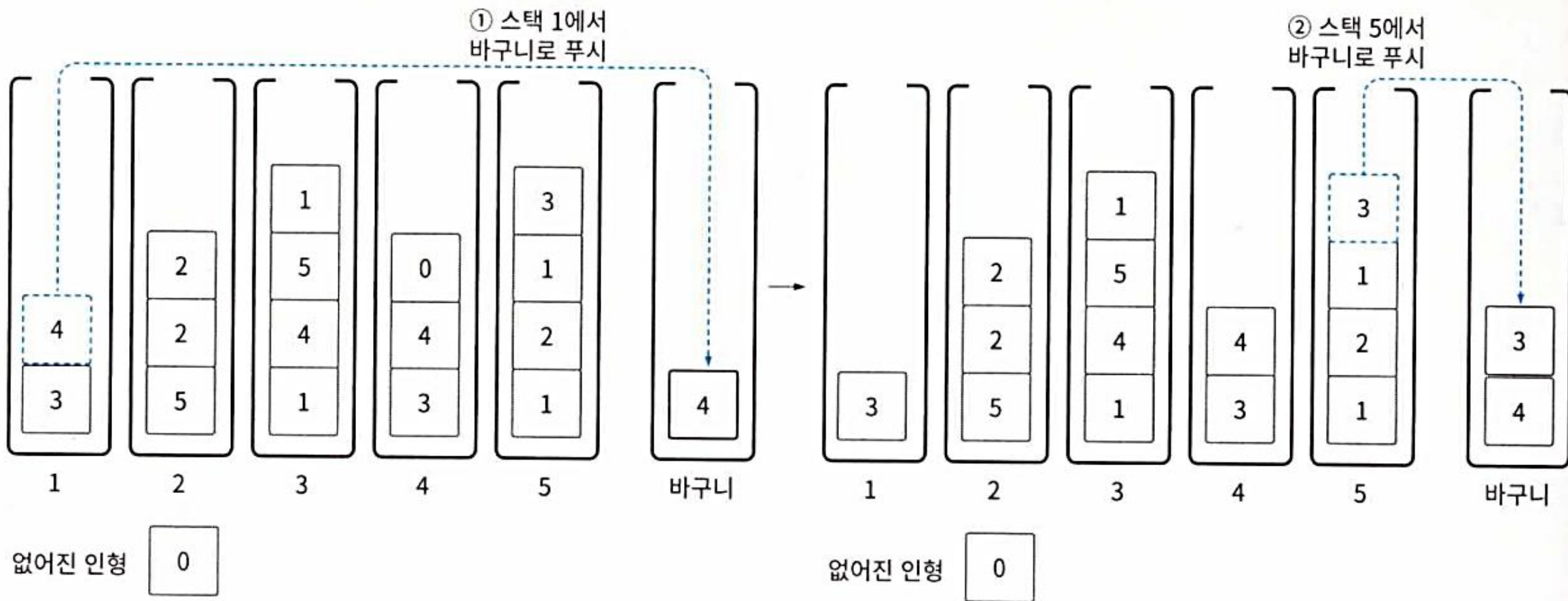
- 두 값이 서로 동일한 경우에 수행되는 동작이다.
- 값 doll을 bucket에 push하지 않으며, bucket 내 top 요소를 pop한다.

**else bucket.pop();**

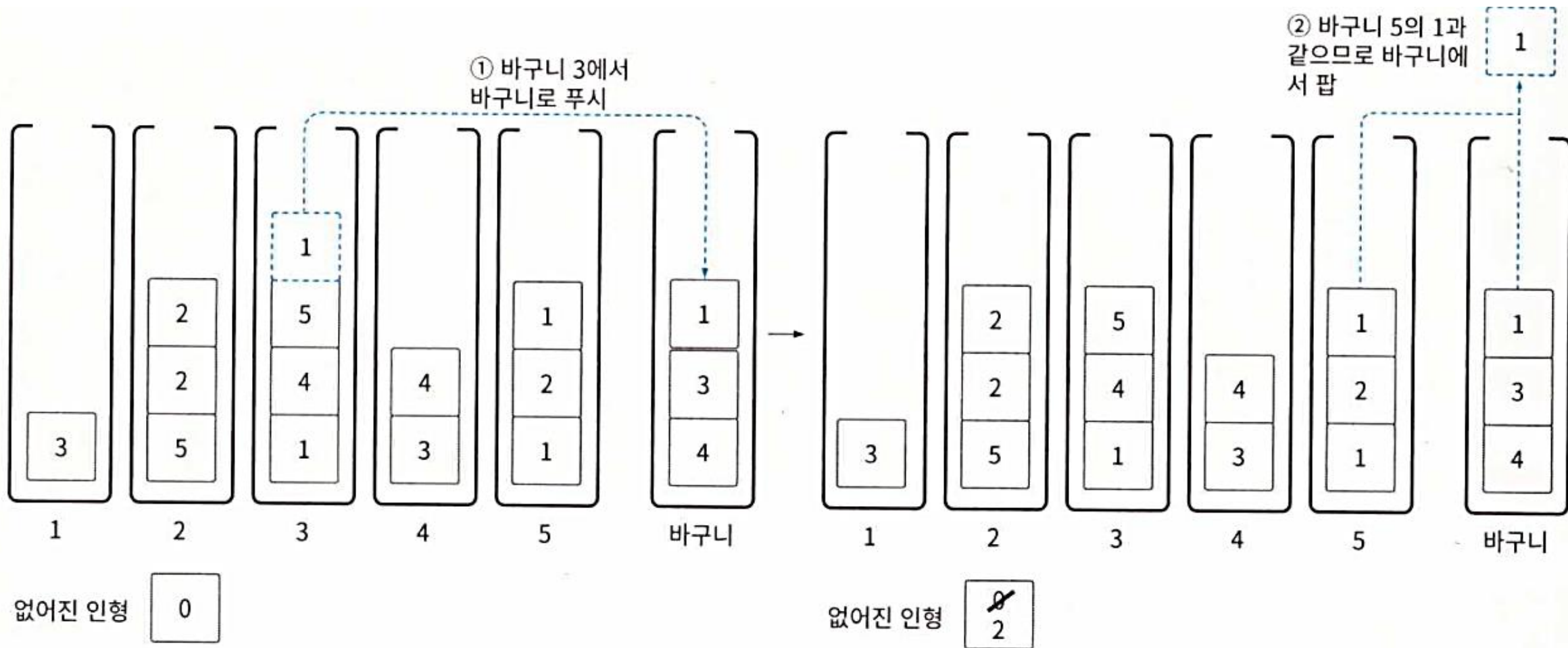
## 문제 3. 문제 해결 방법 기술

- 앞서 수립한 문제 해결 전략을 바탕으로, 문제 해결 방법을 의사코드로 기술한다.
  - ①  $m \leftarrow$  입력 리스트 moves 내 위치한 요소
  - ② 상자의 m번째 열에서 인형을 하나 꺼낸다.
  - ③ 바구니가 빈 경우 꺼낸 인형을 바구니에 추가한다.
  - ④ 바구니가 비지 않은 경우
    - 바구니의 최상단에 위치한 인형과 지금 추가하려는 인형이 같은지 비교한다.
      - 두 인형이 같다면, 바구니 최상단에 있는 인형을 제거하고, 제거된 인형의 개수를 카운트한다.
      - 두 인형이 다르다면, 방금 꺼낸 인형을 바구니에 추가한다.
  - ⑤ 마지막에 스택의 길이를 출력한다.
  - ⑥ 입력 리스트 moves의 모든 요소를 참조할 때 까지 위 단계 ① - ⑤를 반복한다.

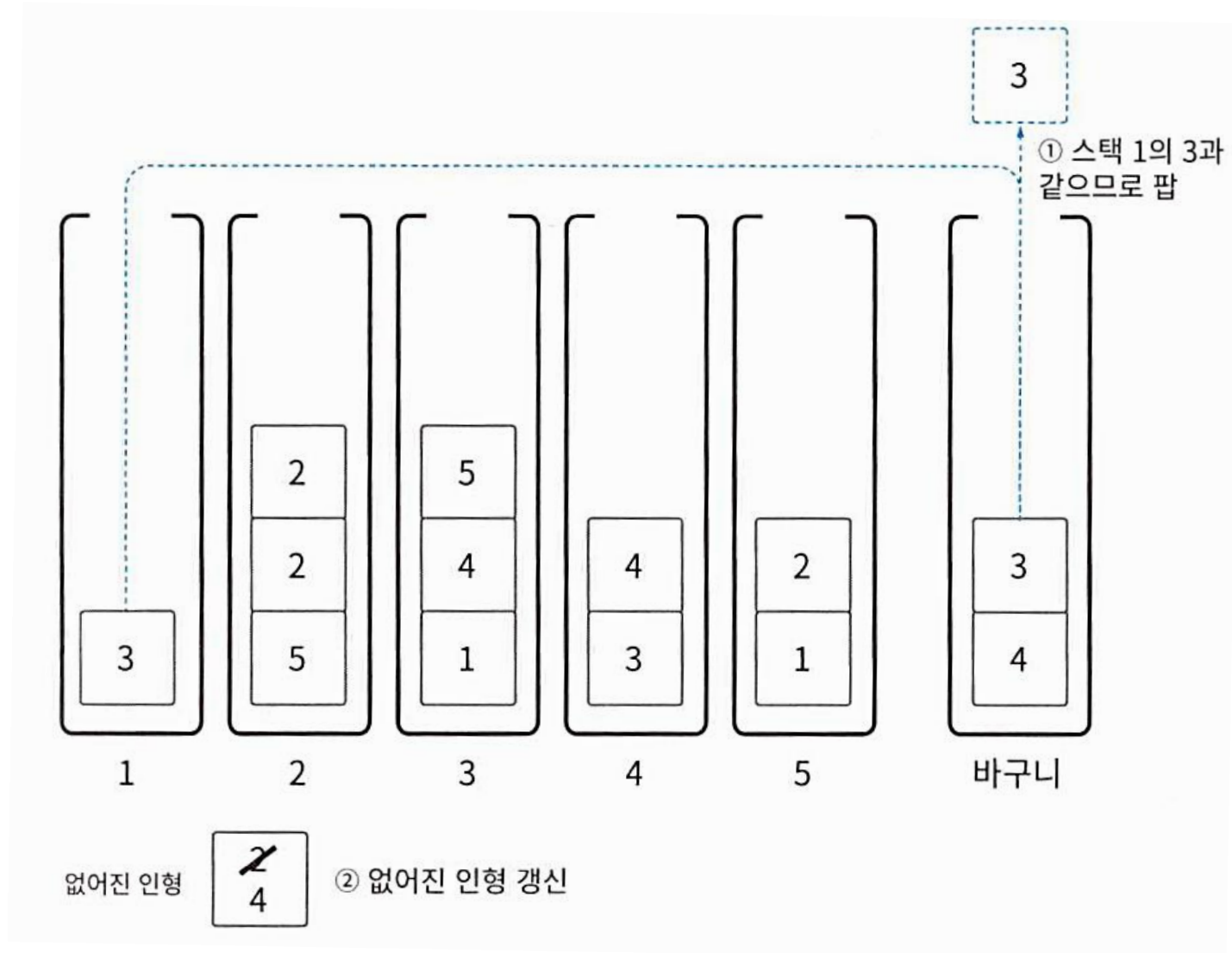
# 문제 3. 수행 예제



# 문제 3. 수행 예제



# 문제 3. 수행 예제



# 문제 3. 구현 및 검증

```
def solution3(board, moves):
    cols = [[] for _ in range(len(board[0]))]

    # board의 각 열을 스택으로 변환한다.
    for i in range(len(board)-1, -1, -1):
        for j in range(len(board[0])):
            if board[i][j]:
                cols[j].append(board[i][j]) # stack's push operation

    bucket = []
    answer = 0 # 사라진 인형의 개수

    # moves 리스트를 참조하여 인형을 bucket으로 옮긴다.
    for m in moves:
        if cols[m-1]:
            doll = cols[m-1].pop()

            if bucket and bucket[-1] == doll:
                bucket.pop()
                answer += 2
            else:
                bucket.append(doll)

    return answer
```

## 테스트 케이스

```
# Test
board = [[0,0,0,0,0],[0,0,1,0,3],[0,2,5,0,1],
          [4,2,4,4,2],[3,5,1,3,1]]
moves = [1,5,3,5,1,2,1,4]

answer = solution4(board, moves)
print(answer)
```



# 연습문제 4. 요세푸스 알고리즘

## 문제 4. 요세푸스 알고리즘

제한시간: 20분

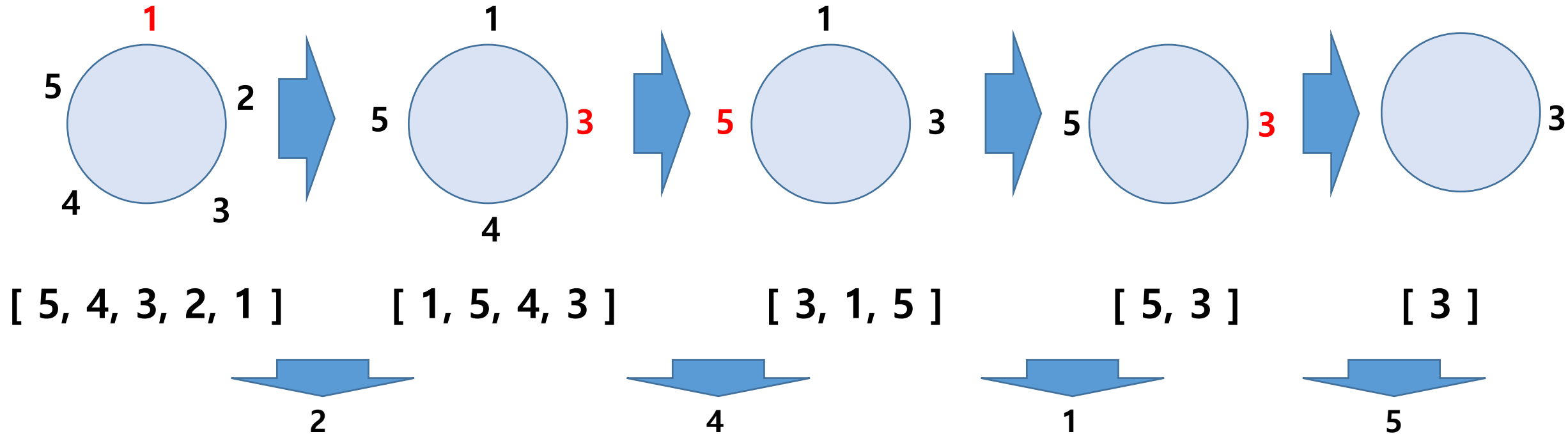
- 1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수  $K(\leq N)$ 가 주어진다.
- 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다.
- 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은  $\langle 3, 6, 2, 7, 5, 1, 4 \rangle$ 이다.
- 이때, N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 **solution4 함수**를 작성하시오.
  - **예제 1:** [입력]  $N = 7, K = 3$   
[출력] 3, 6, 2, 7, 5, 1, 4
  - **예제 2:** [입력]  $N = 5, K = 2$   
[출력] 2, 4, 1, 5, 3

## 문제 4. 문제 분석

- 문제에 키워드가 숨어있다
  - “1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고 ...” → 원순열
- 삭제되는 K번째 숫자는 기준 위치의 숫자를 포함한 K번째 숫자임.
- 기준 위치는 1번부터 시작하고, 특정 숫자가 삭제되면 다음 기준 위치는 삭제된 숫자 바로 오른쪽에 위치한 숫자로 새로 지정.
- 나머지 숫자들은 삭제되지 않고 원래의 자리를 그대로 유지함. 즉, K번째 수가 삭제된 후에도 삭제되지 않은 나머지 수들의 순서는 변동이 없음.

# 문제 4. 문제 분석 (직접 풀어보기)

(예)  $N=5, K=2$



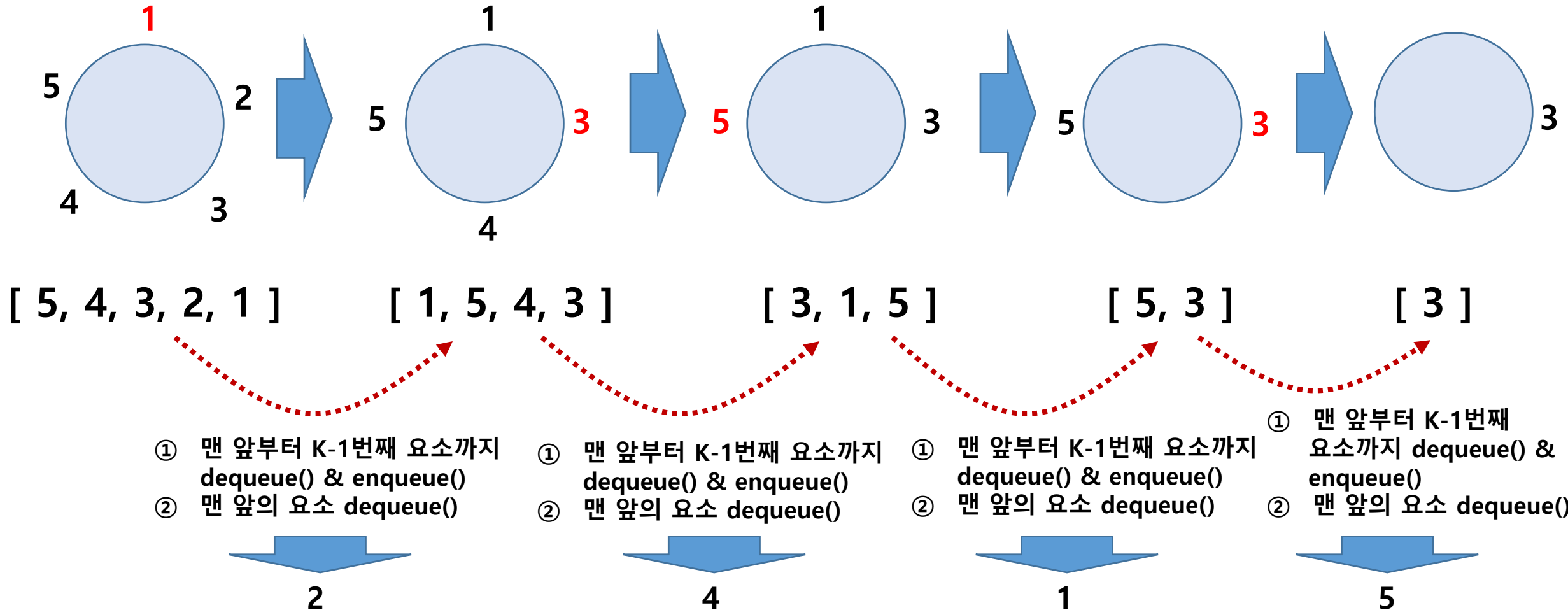
최종 반환 결과 :  $[2\ 4\ 1\ 5\ 3]$

## 문제 4. 문제 해결 전략 수립

- 앞에서부터 순차적으로 반복 처리하고, 삭제되는 숫자 이외의 나머지 수는 그 순서가 그대로 유지되므로, 원형 큐를 이용 수 있다.
- 단, 파이썬으로 구현하는 경우 일반 큐를 이용해도 메모리 효율성 측면에서 무방하다.

# 문제 4. 문제 해결 전략 수립

(예)  $N=5, K=2$



## 문제 4. 구현 및 검증

### 테스트 케이스

```
1 # Example test (N=5, K=2)
2 print( solution4(5, 2) )
3
4 # Example test (N=7, K=3)
5 print( solution4(7, 3) )
```

```
[2, 4, 1, 5, 3]
[3, 6, 2, 7, 5, 1, 4]
```

```
import queue
```

```
def solution4(N, K):
    q = queue.Queue()
    results = []
```

```
# 큐에 1부터 N까지 삽입
```

```
for i in range(1, N + 1):
    q.put(i)
```

```
# 큐를 이용하여 요세푸스 알고리즘 해결하기
```

```
while q.qsize() > 1:
    for _ in range(K-1):
        q.put(q.get())
    results.append(q.get())
```

```
results.append(q.get())
```

```
return results
```

# Q & A