

# 특강 1일차 - 자료구조와 알고리즘

# Contents

<b>01</b>	—————	코딩테스트 소개 및 준비 방법
<b>02</b>	—————	알고리즘
<b>03</b>	—————	선형 자료구조
<b>04</b>	—————	리스트
<b>05</b>	—————	연결 리스트
<b>06</b>	—————	스택
<b>07</b>	—————	큐
<b>08</b>	—————	덱

# 코딩 테스트 소개 및 준비 방법

---

## 코딩 테스트

- SW직군에 지원하는 대상자들의 SW 문제 해결 역량을 측정하기 위한 테스트
- SW 문제에 대한 이해와 코딩까지의 종합적인 SW 문제해결 역량을 평가하는 데 초점을 맞춤
- 문제 유형
  - 알고리즘, 자료구조, 수학, 문자열 처리 등
- 제한 조건
  - 제한된 시간 안에 문제를 해결하고, 특정 메모리 범위 내에서 작동해야 함
- 프로그래밍 언어
  - Python, Java, C++, JavaScript 등 다양한 언어 지원

# 코딩 테스트 준비

- 자료구조
  - List, Set, Map
  - Stack
  - Queue, Priority Queue
  - Graph

# 코딩 테스트 준비

- 알고리즘
  - 완전 탐색
  - 조합적 문제 해결(순열 / 조합 / 부분집합)
  - 백트래킹
  - 탐욕 알고리즘
  - 서로소 집합(Union Find 알고리즘)
  - 그래프 관련 알고리즘(탐색-BFS/DFS, 최소 신장 트리, 최단 경로)
  - 분할 정복 알고리즘 (이진 탐색)
  - DP
  - 문자열

## 코딩 테스트 준비

- 활용 가능한 온라인 저지 사이트에서 다양한 문제 풀어보기
  - 백준
  - SWEA
  - 코드트리
  - 정올

# 코딩 테스트 준비

- 초급

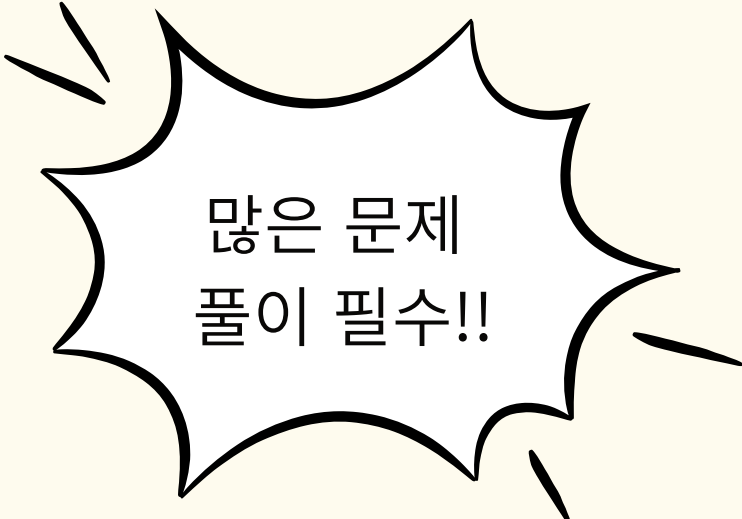
- 백준 단계별로 문제 풀어보기
- Solved.ac Class1~3까지 풀어보기(기본기 위주)
- 기본 구현 및 자료구조 활용, DFS/BFS 위주

- 중급

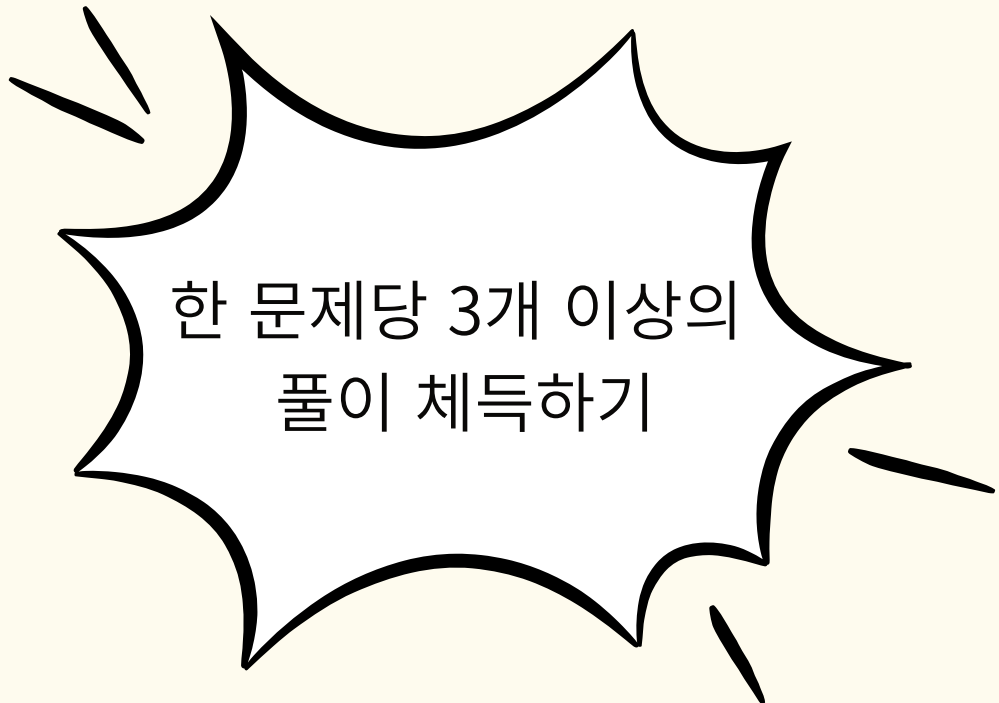
- 백준 골드 티어의 다양한 문제 풀어보기
- 초급 + 그래프이론 + 다양한 알고리즘

- 고급

- 백준 플래티넘의 다양한 문제 풀어보기
- 중급 + 최적화



많은 문제  
풀이 필수!!



한 문제당 3개 이상의  
풀이 체득하기



# 알고리즘

---

# 알고리즘

- 문제를 해결하기 위해 필요한 절차나 방법을 명확하고 구체적으로 기술한 것
- 알고리즘은 유한한 성질을 가짐
- "알고리즘"이라는 용어는 페르시아 수학자 알-후아리즈미(Al-Khwarizmi)의 이름에서 유래

1부터 100까지의 숫자 중 홀수만 모두 더한다면?

$$1+3+5+7+9+ \dots + 99$$

# 알고리즘의 표현 방법

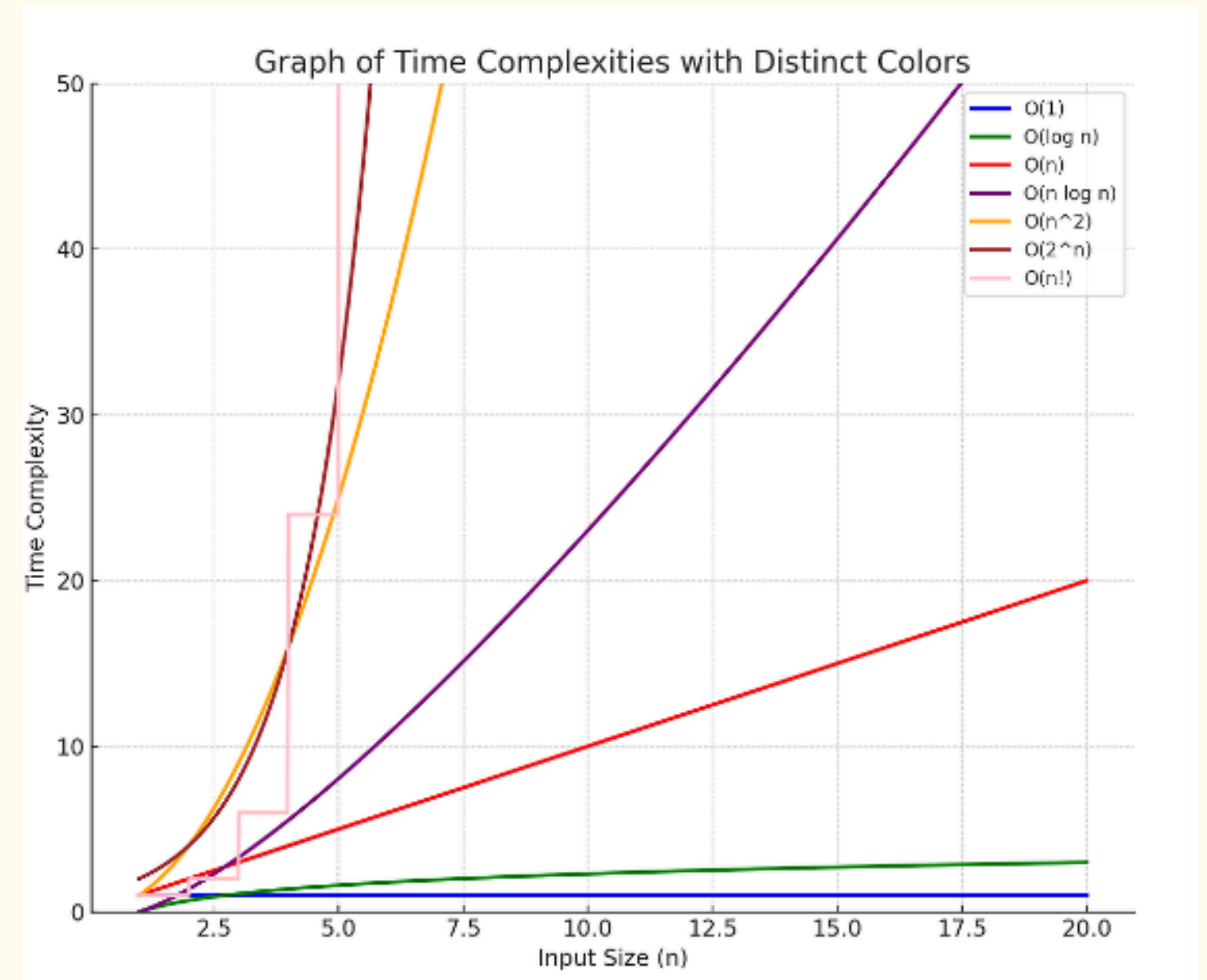
- 의사 코드(Pseudocode)
  - 코드처럼 보이지만 실제 프로그래밍 언어는 아닌 방식으로 알고리즘을 설명
- 순서도(Flowchart)
  - 알고리즘의 흐름을 시각적으로 표현
- 자연어
  - 일상 언어를 사용하여 단계별로 설명
- 프로그래밍 언어
  - 실제 코드를 통해 알고리즘 구현

## 알고리즘의 효율성

- 시간 복잡도(Time Complexity)
  - 알고리즘이 실행되는 데 걸리는 시간의 효율성을 나타내는 척도
  - 입력 크기에 따라 알고리즘의 수행 시간이 어떻게 변하는지를 분석
- 공간 복잡도(Space Complexity)
  - 알고리즘이 실행되는 동안 필요한 메모리 공간의 효율성을 나타내는 척도
  - 입력 크기에 따라 알고리즘이 차지하는 메모리 공간이 어떻게 변하는지를 분석

# 알고리즘의 효율성

- 최선의 경우(Best Case)
  - 빅오메가표기법 사용( $\Omega(n)$ )
  - 최선의 시나리오로 최소 이만한 시간이 걸림
- 최악의 경우(Worst Case)
  - 빅오표기법 사용( $O(n)$ )
  - 최악의 시나리오로 아무리 오래 걸려도 이 시간보다 덜 걸림
- 평균적인 경우(Average Case)
  - 빅세타표기법 사용( $\Theta(n)$ )
  - 평균시간을 나타냄



## 알고리즘의 효율성

- 빅-오( $O$ ) 표기법
  - 시간 복잡도 함수 중에서 가장 큰 영향력을 주는  $n$ 에 대한 항만을 표시하는 점근적 표기법
  - 계수(Coefficient)는 생략
- $O(2n^2 + 10n)$ 
  - $O(n^2)$

## 알고리즘의 효율성

- 정렬되지 않은 N개의 서로 다른 값의 배열에서 특정 값을 탐색하는 경우의 시간 복잡도를 생각한다면?
  - 최선
  - 최악
  - 평균
- 정렬되어 있는 경우는?

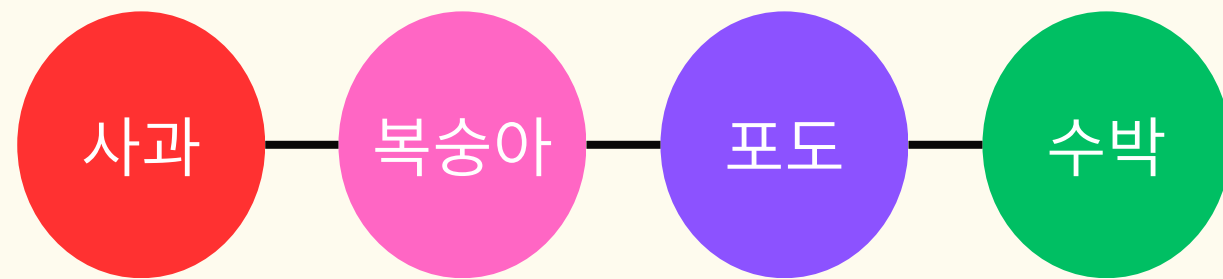
# 선형 자료구조

---



## 선형 자료구조

- 하나의 자료 뒤에 하나의 자료가 존재하는 자료들 간의 관계 차수가 1:1인 관계
- 배열, 리스트, 스택, 큐, 덱



# 리스트(List)

---

# 리스트

- 순서를 가진 데이터의 집합, 목록
- 데이터의 중복 허용
- 순차 리스트
  - 논리적인 순서와 물리적인 순서가 일치
  - 배열
- 연결 리스트
  - 논리적인 순서와 물리적인 순서가 다를 수 있음
  - 동적 메모리 할당

# 리스트

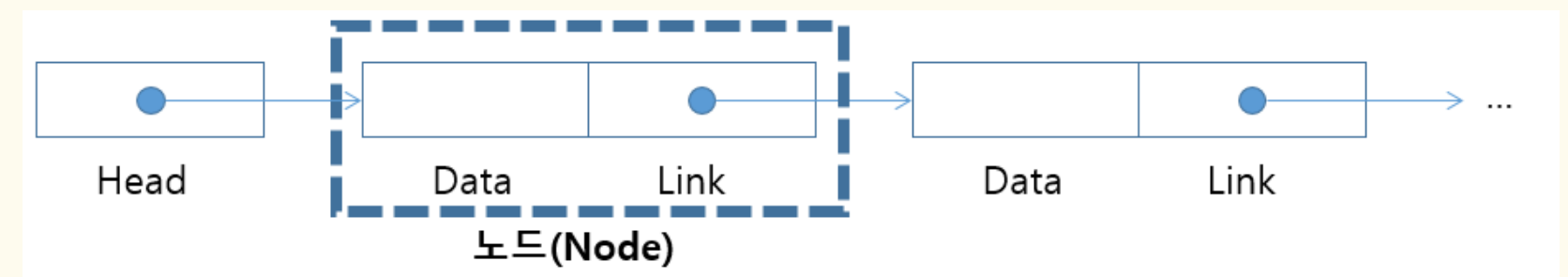
- 1차원 배열 활용
  - 배열 탐색
  - 왼쪽, 오른쪽으로 밀기
  - 순환
- 2차원 배열 활용
  - 배열 탐색(행 우선, 열 우선)
  - 델타 이용한 방향 탐색
  - 회전(시계, 반시계)
  - 좌우, 상하 뒤집기 등

# 연결 리스트(Linked List)

---

# 연결 리스트(Linked List)

- 연결 리스트는 노드(Node)들이 포인터를 통해 연결된 선형 자료구조
- 각 노드는 데이터와 다음 노드를 가리키는 포인터를 포함
- 특징
  - 크기가 동적이며 노드의 삽입/삭제가 용이
  - 메모리 낭비가 적고, 요소의 물리적 위치가 연속적이지 않음
  - 임의 접근이 불가능하며, 탐색 속도가 배열에 비해 느림
- 활용
  - 메모리 관리를 효율적으로 해야 하는 경우
  - 데이터의 삽입/삭제가 빈번히 일어나는 경우



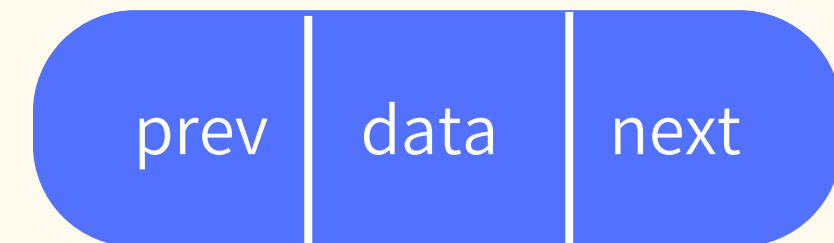
# 연결 리스트(Linked List)

- 종류

- 단순 연결 리스트

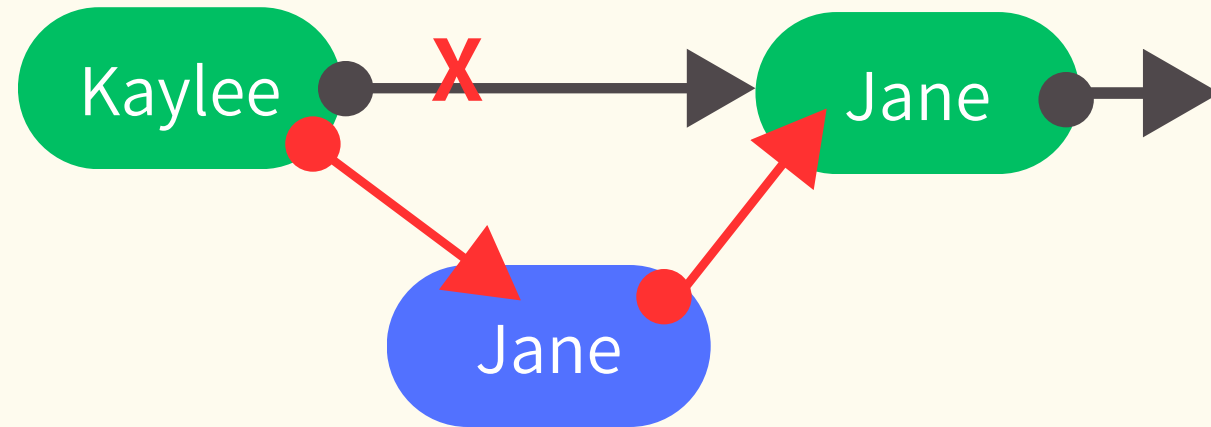


- 이중 연결 리스트

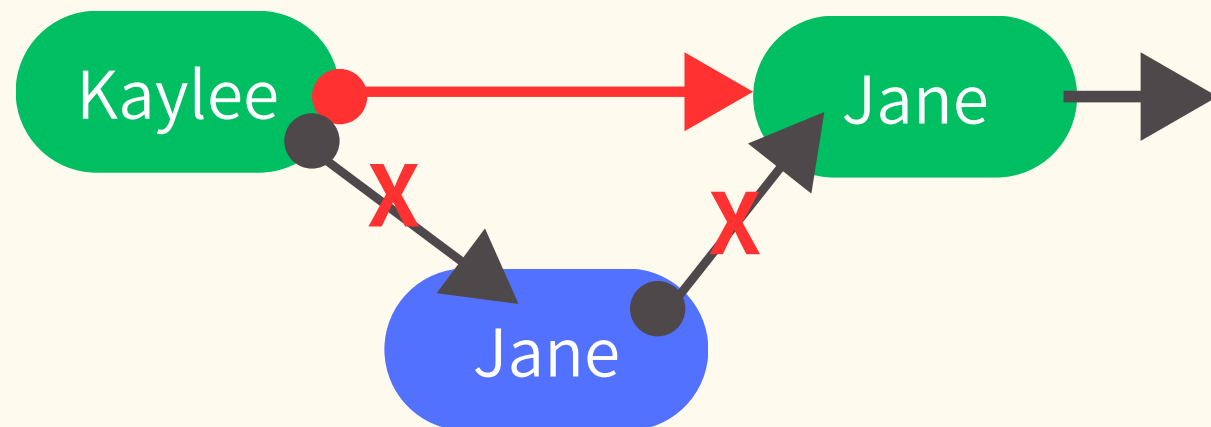


## 단순 연결 리스트

- 원소 삽입



- 원소 삭제





# 스택(Stack)

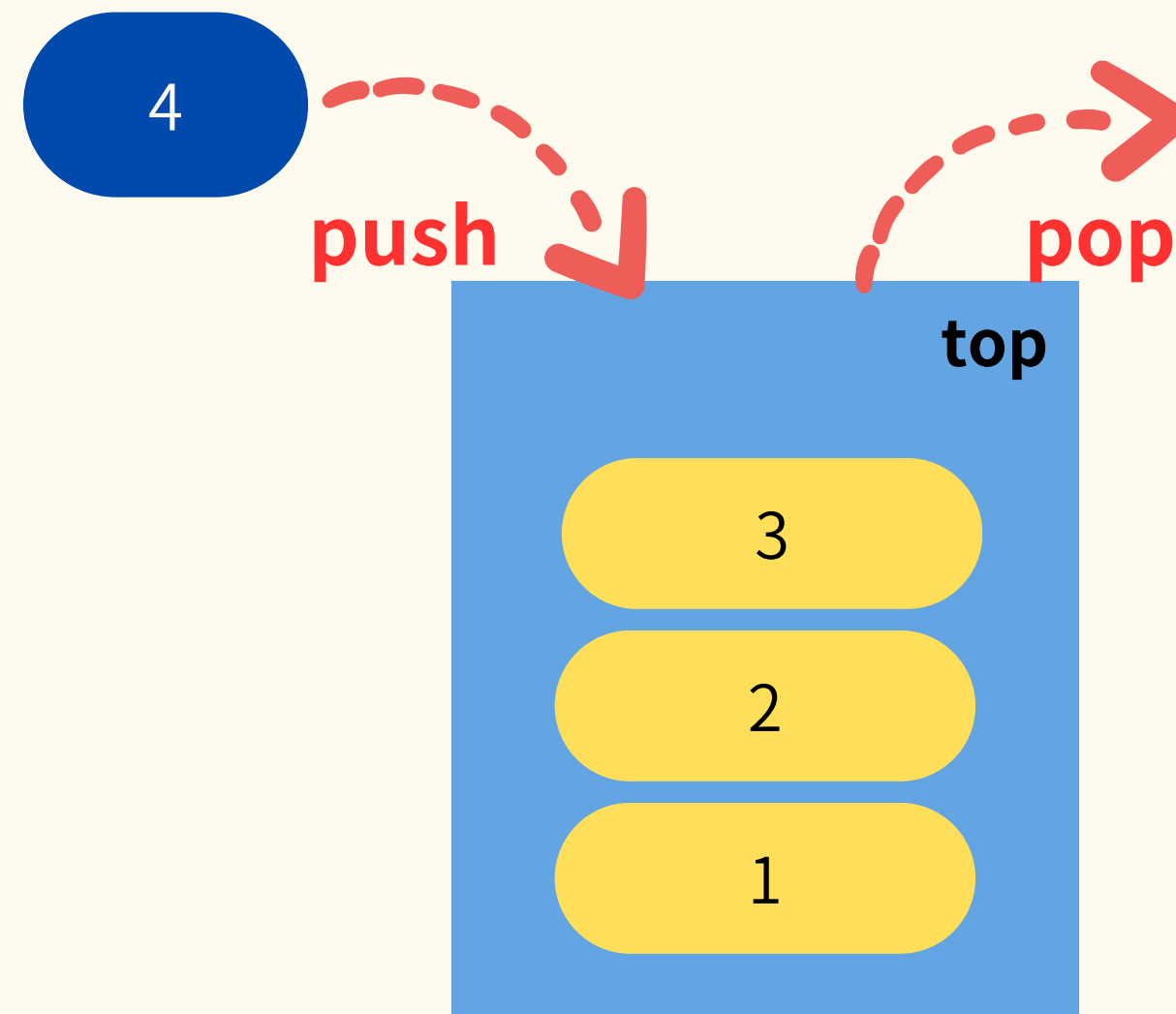
---

# 스택(Stack)

- 데이터를 차곡 차곡 쌓아 올린 형태의 자료구조
- 후입선출(Last In First Out) 구조
- 주요 연산
  - Push : 데이터 추가
  - Pop : 데이터 삭제
  - Peek : 데이터 확인

## stack ★

1. 명사 (보통 깔끔하게 정돈된) 무더기[더미] (→haystack)
2. 명사 많음, 다량
3. 동사 (깔끔하게 정돈하여) 쌓다[포개다]; 쌓이다, 포개지다
4. 동사 (어떤 곳에 물건을 쌓아서) 채우다



# 스택(Stack)

- java.util.Stack<E>
- 주요 기능
  - push()
  - pop()
  - peek()
  - isEmpty()
  - size()

## 스택의 활용

- 괄호 검사
- `if((i==0)|| (j==0))`
- 활용 문제
  - 괄호
    - <https://www.acmicpc.net/problem/9012>

## 스택의 활용

- 후위 표기식을 이용한 계산
- $2 * (4+6) / 10 - 8$
- 활용 문제
  - 후위표기식2
    - <https://www.acmicpc.net/problem/1935>

## 스택의 활용

- 브라우저의 뒤로가기, 앞으로가기 구현
- 활용 문제
  - 브라우저
    - <https://jungol.co.kr/problem/1015>

# 큐(Queue)

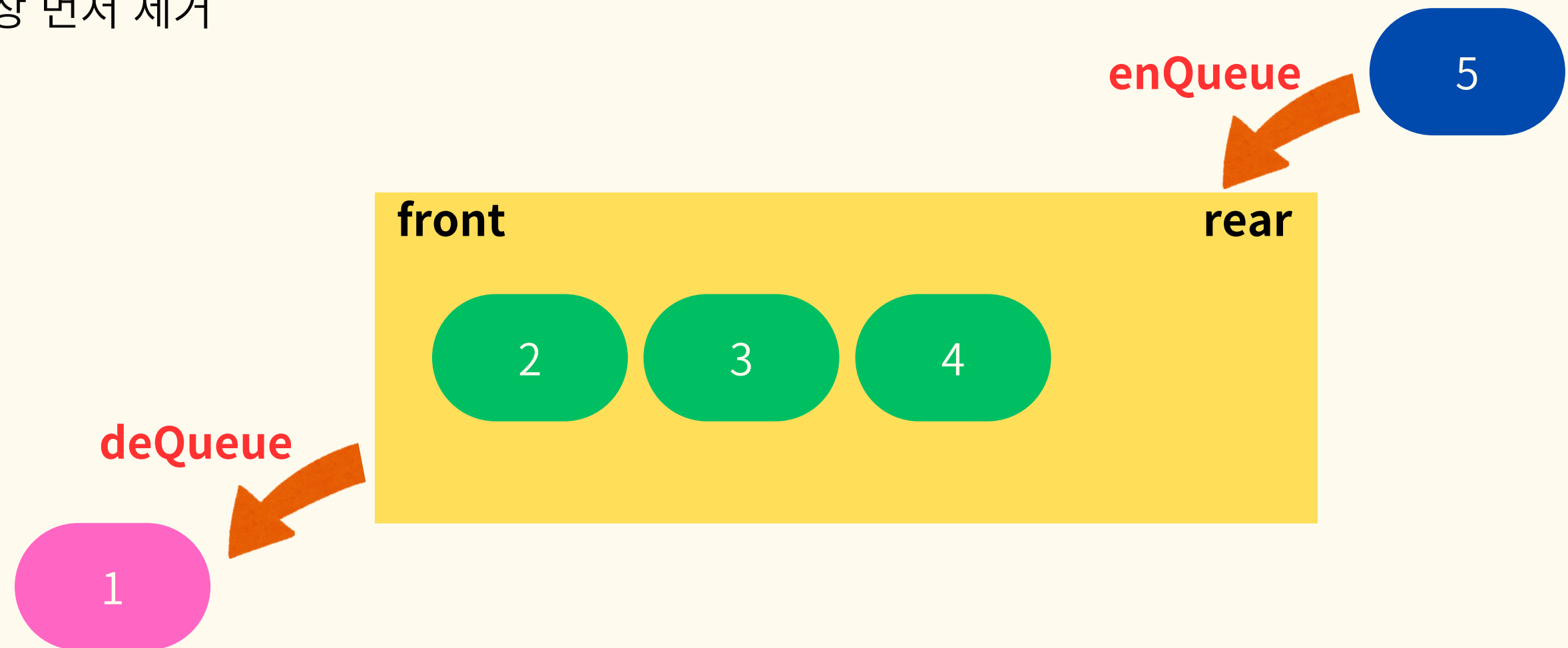
---

# 큐(Queue)

- 대기열
- 선입선출(First In First Out) 구조의 자료구조
- 가장 먼저 삽입된 데이터가 가장 먼저 제거
- 주요 연산
  - enqueue
  - dequeue
  - peek

## queue ★

1. 명사 (무엇을 기다리는 사람·자동차 등의) 줄
2. 명사 큐, 대기 행렬
3. 동사 줄을 서서 기다리다
4. 동사 (수행할 업무들이) 대기 행렬을 만들다[이루다]





# 큐(Queue)

- `java.util.Queue<E>`
- 주요 기능
  - `offer()`
  - `poll()`
  - `peek()`
  - `isEmpty()`
  - `size()`

## 큐의 활용

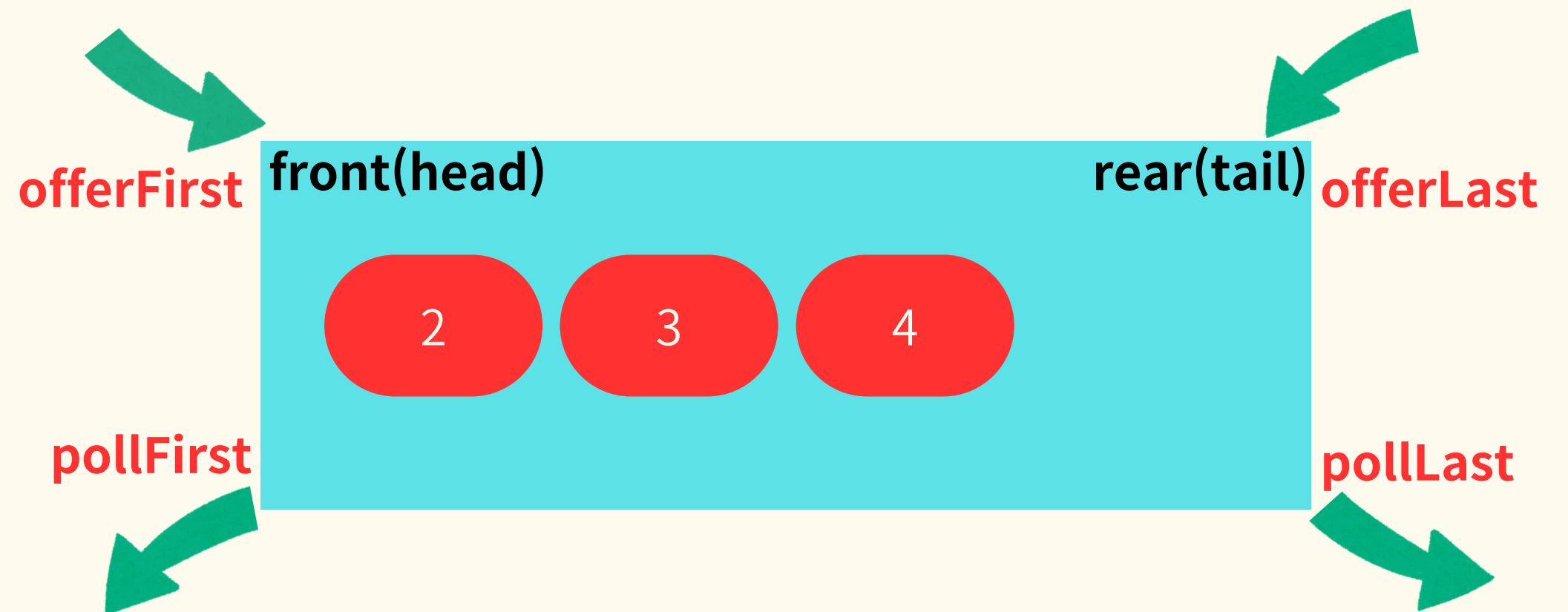
- 대기열과 같은 우선순위 작업 관리
- BFS
- 활용 문제
  - 프린터 큐
    - <https://www.acmicpc.net/problem/1966>

# 덱(Deque)

---

## 덱(Deque)

- double-ended queue
- 양방향으로 삽입과 삭제가 가능
- 큐는 한쪽 끝에서 삽입, 반대쪽 끝에서 삭제를 하는 반면, 덱은 양쪽 끝에서 모두 삽입과 삭제
- 큐(Queue)와 스택(Stack)의 기능을 모두 사용할 수 있음
- FIFO(First-In, First-Out) 가능
- LIFO(Last-In, First-Out) 가능



# 덱(Deque)

- java.util.Deque<E>
- 주요 기능
  - offerFirst(), offerLast()
  - pollFirst(), pollLast()
  - peekFirst(), peekLast()
  - isEmpty()
  - size()

Summary of Deque methods

	First Element (Head)		Last Element (Tail)	
	<i>Throws exception</i>	<i>Special value</i>	<i>Throws exception</i>	<i>Special value</i>
Insert	<code>addFirst(e)</code>	<code>offerFirst(e)</code>	<code>addLast(e)</code>	<code>offerLast(e)</code>
Remove	<code>removeFirst()</code>	<code>pollFirst()</code>	<code>removeLast()</code>	<code>pollLast()</code>
Examine	<code>getFirst()</code>	<code>peekFirst()</code>	<code>getLast()</code>	<code>peekLast()</code>

## 덱의 활용

- 회문(팰린드롬) 검사
- 회전 큐(Rotation Queue)
- 스택 또는 큐로 사용
- 활용 문제
  - 팰린드롬
    - <https://www.acmicpc.net/problem/10174>
  - 회전하는 큐
    - <https://www.acmicpc.net/problem/1021>

Thank you!

---