SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

APS 기본

SAMSUNG SW ACADEMY FOR YOUTH

APS(Algorithm Problem Solving)

링크드리스트 (Linked List)

- 연결 스택 / 연결 큐
- 요세푸스 문제
- 삽입정렬

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

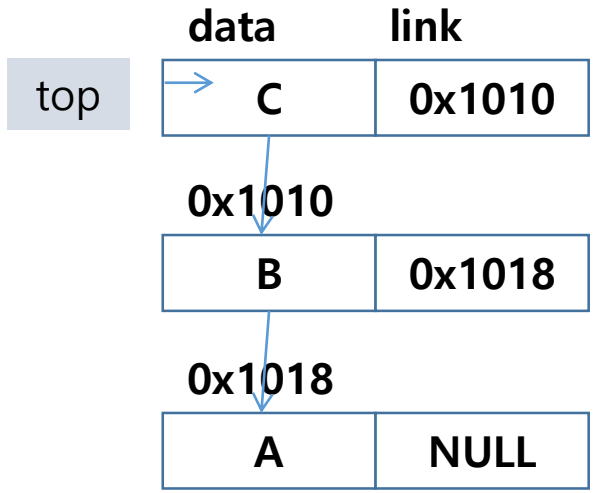
연결 스택
(Linked Stack)

리스트를 이용한 스택

- 리스트를 이용해 스택을 구현할 수 있다.

- 스택의 원소 : 리스트의 노드
  - 스택 내의 순서는 리스트의 링크를 통해 연결됨
    - Push : 리스트의 마지막에 노드 삽입
    - Pop : 리스트의 마지막 노드 반환/삭제

- 변수 top
  - 리스트의 마지막 노드를 가리키는 변수
  - 초기 상태 : top = null

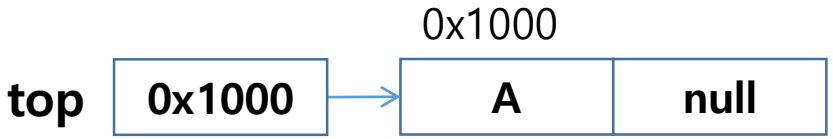


리스트를 이용한 스택

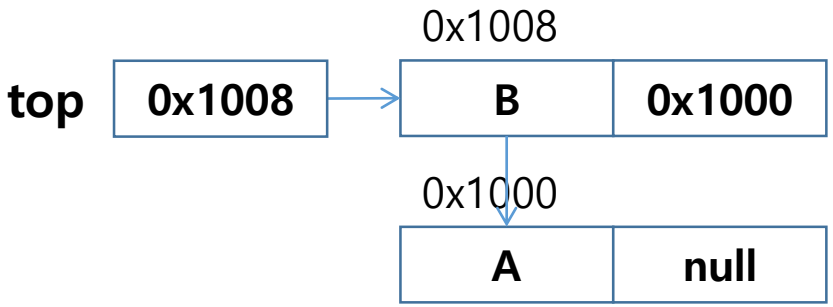
리스트를 이용해 Push와 Pop 연산 구현

① null 값을 가지는 노드를 만들어 스택 초기화

② 원소 A 삽입 : push(A)

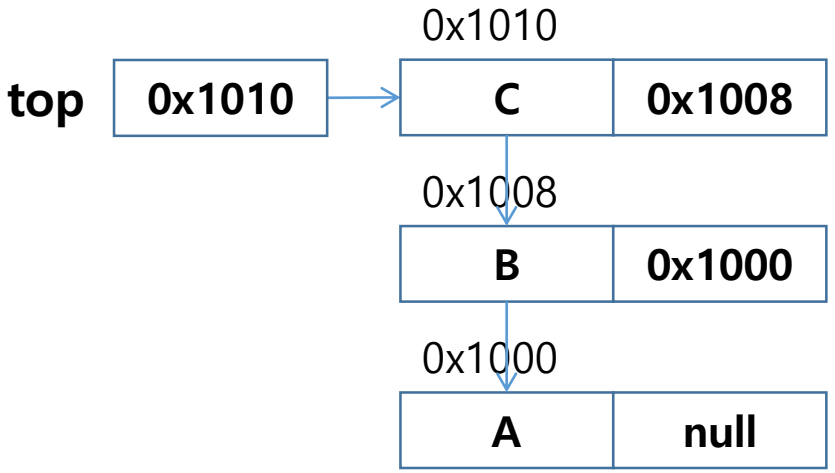


③ 원소 B 삽입 : push(B)

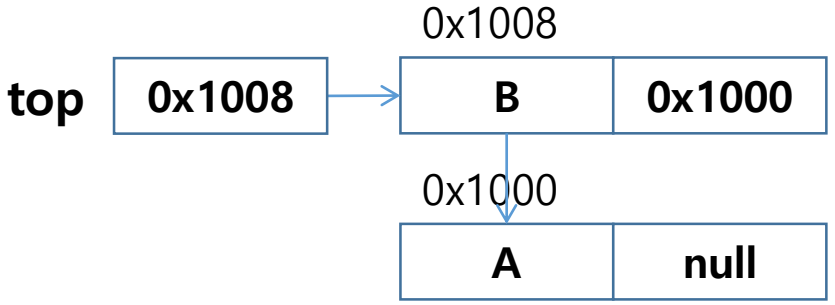


리스트를 이용한 스택

④ 원소 C 삽입 : Push(C)



⑤ 원소 반환 : Pop



리스트를 이용한 스택

Push/Pop

```
push(i) {                    //원소 i를 스택에 push
    temp = createNode();
    temp.data = i;
    temp.link = top;
    top = temp;
}

pop() {
    temp = top
    if (top == null) return 0;
    else {
        item = temp.data;
        top = temp.link  // top이 가리키는 노드 변경
        free(temp);
        return item;
    }
}
```

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

연결 큐
(Linked Queue)

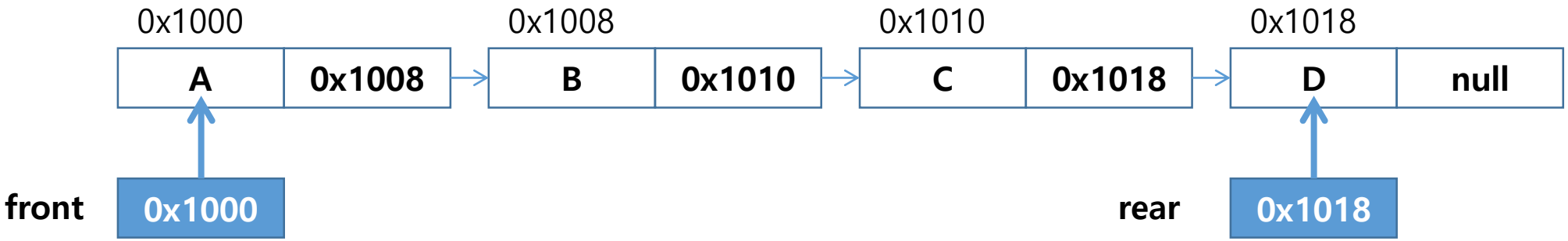
연결 큐의 구조

단순 연결 리스트(Linked List)를 이용한 큐

- 큐의 원소 : 단순 연결 리스트의 노드
- 큐의 원소 순서 : 노드의 연결 순서. 링크로 연결되어 있음
- front : 첫 번째 노드를 가리키는 링크
- rear : 마지막 노드를 가리키는 링크

상태 표현

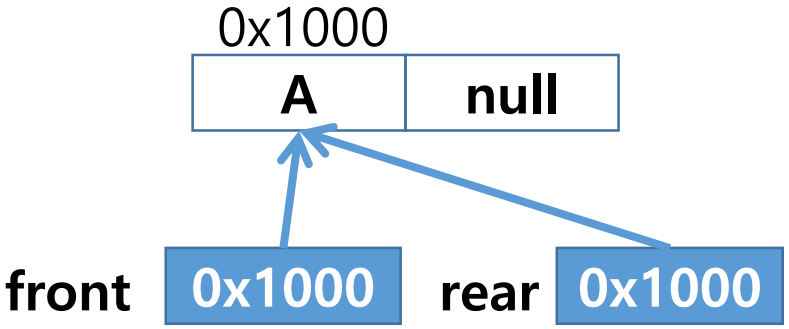
- 초기 상태 : front = rear = null
- 공백 상태 : front = rear = null



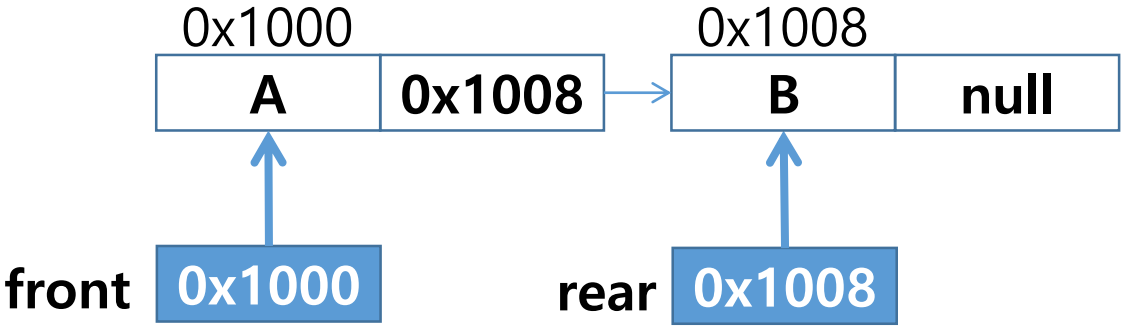
연결 큐의 연산 과정

1) 공백 큐 생성 : createLinkedQueue();

2) 원소 A 삽입 : enQueue(A);

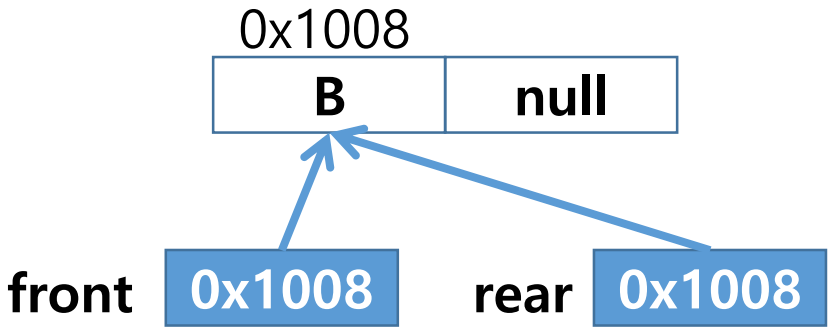


3) 원소 B 삽입 : enQueue(B);

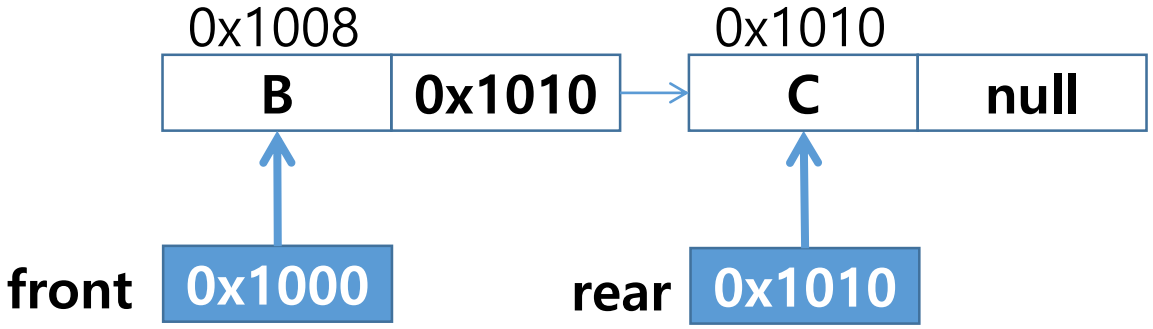


연결 큐의 연산 과정

4) 원소 삭제 : deQueue();

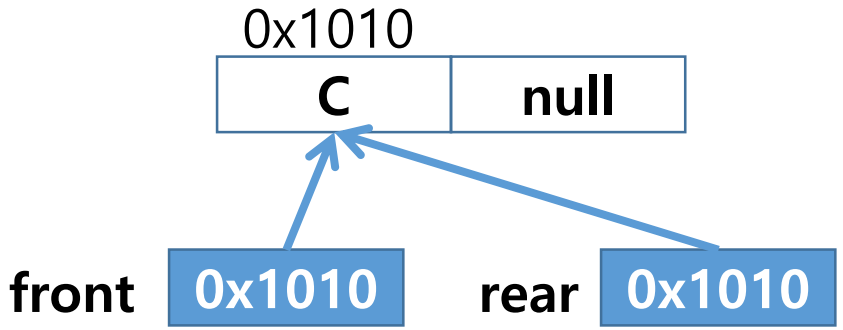


5) 원소 C 삽입 : enQueue(C);



연결 큐의 연산 과정

6) 원소 삭제 : deQueue();



7) 원소 삭제 : deQueue();

연결 큐의 구현

초기 공백 큐 생성 : createLinkedQueue()

- 리스트 노드 없이 포인터 변수만 생성함
- front와 rear를 null로 초기화

```
createLinekdQueue() {
    front ← null;
    rear ← null;
}
```

공백상태 검사 : isEmpty()

- 공백상태 : front = rear = null

```
isEmpty() {
    if(front == null) return true;
    else return false;
}
```

연결 큐의 구현

삽입 : enQueue(item)

① 새로운 노드 생성 후 데이터 필드에 item 저장

② 연결 큐가 공백인 경우, 아닌 경우에 따라 front, rear 변수 지정

```
enQueue(item) {
    new ← getNode();    //getNode() : 새로운 노드 할당 후 리턴
    new.data ← item;
    new.link ← null;
    if(front == null) {
        rear ← new;
        front ← new;
    } else {
        rear.link ← new;
        rear ← new;
    }
}
```

연결 큐의 구현

삭제 : deQueue()

① old가 지울 노드를 가리키게 하고, front 재설정

② 삭제 후 공백 큐가 되는 경우, rear도 null로 설정

③ old가 가리키는 노드 삭제하고 메모리 반환

```
deQueue() {
    if(isEmpty()) Queue_Empty();
    else {
        old ← front;
        item ← front.data;
        front ← front.link;
        if (isEmpty()) rear ← NULL;
        free(old);
        return item;
    }
}
```

우선순위 큐

우선순위 큐의 구현

- 배열을 이용한 우선순위 큐
- 리스트를 이용한 우선순위 큐

우선순위 큐의 기본 연산

- 삽입 : enQueue
- 삭제 : deQueue

배열을 이용한 우선순위 큐

배열을 이용하여 우선순위 큐 구현

- 배열을 이용하여 자료 저장
- 원소를 삽입하는 과정에서 우선순위를 비교하여 적절한 위치에 삽입하는 구조
- 가장 앞에 최고 우선순위의 원소가 위치하게 됨

문제점

- 배열을 사용하므로, 삽입이나 삭제 연산이 일어날 때 원소의 재배치가 발생함
- 이에 소요되는 시간이나 메모리 낭비가 큼

리스트를 이용한 우선순위 큐

리스트를 이용하여 우선순위

- 연결 리스트를 이용하여 자료 저장
- 원소를 삽입하는 과정에서 리스트 내 노드의 원소들과 비교하여 적절한 위치에 노드를 삽입하는 구조
- 리스트의 가장 앞쪽에 최고 우선순위가 위치하게 됨

배열 대비 장점

- 삽입/삭제 연산 이후 원소의 재배치가 필요 없음
- 메모리의 효율적인 사용이 가능함

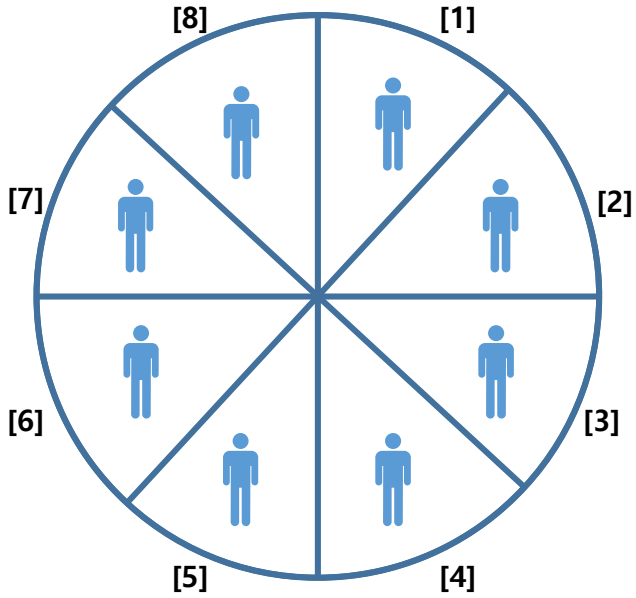
SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

요세푸스 문제
(Josephus Problem)

요세푸스 문제 (Josephus Problem)

- 1 ~ N 까지의 사람이 원을 이루면서 앉아 있다.
- 양의 정수 K (≤N)이 주어지고 순서대로 K번째 사람을 제거한다.
- N명이 모두 제거 될 때까지 반복된다. 제거 되는 순서를 출력하라.



N = 8, K = 3 일때

요세푸스 문제 (Josephus Problem)

Confidential

- 배열의 경우...

요세푸스 문제 (Josephus Problem)

- 첫 번째 병사의 값을 음수 처리 후, 이동.
- 이동 시 병사의 값을 확인하여 count.
  - 음수라면 카운트 하지 않음
  - 양수라면 카운트 +1.
- 음수로 변한 병사의 명수를 저장.
- 남은 병사의 명수가 2이면 남은 병사의 인덱스 값을 확인.

요세푸스 문제 (Josephus Problem)

Confidential

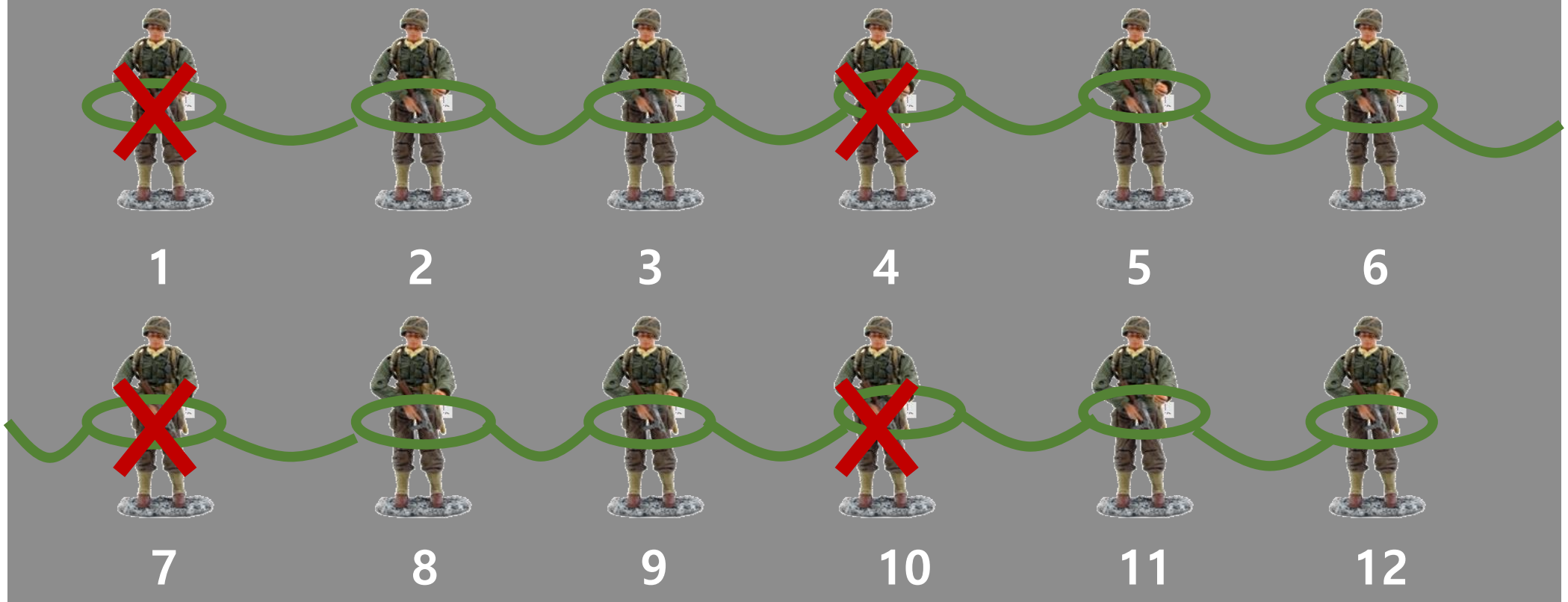
- **배열의 경우...문제점은???**

- **배열 내 순환 횟수 증가.**
  - 인원이 많아질 수록 순환 회수가 커져서 시간이 오래 걸린다.

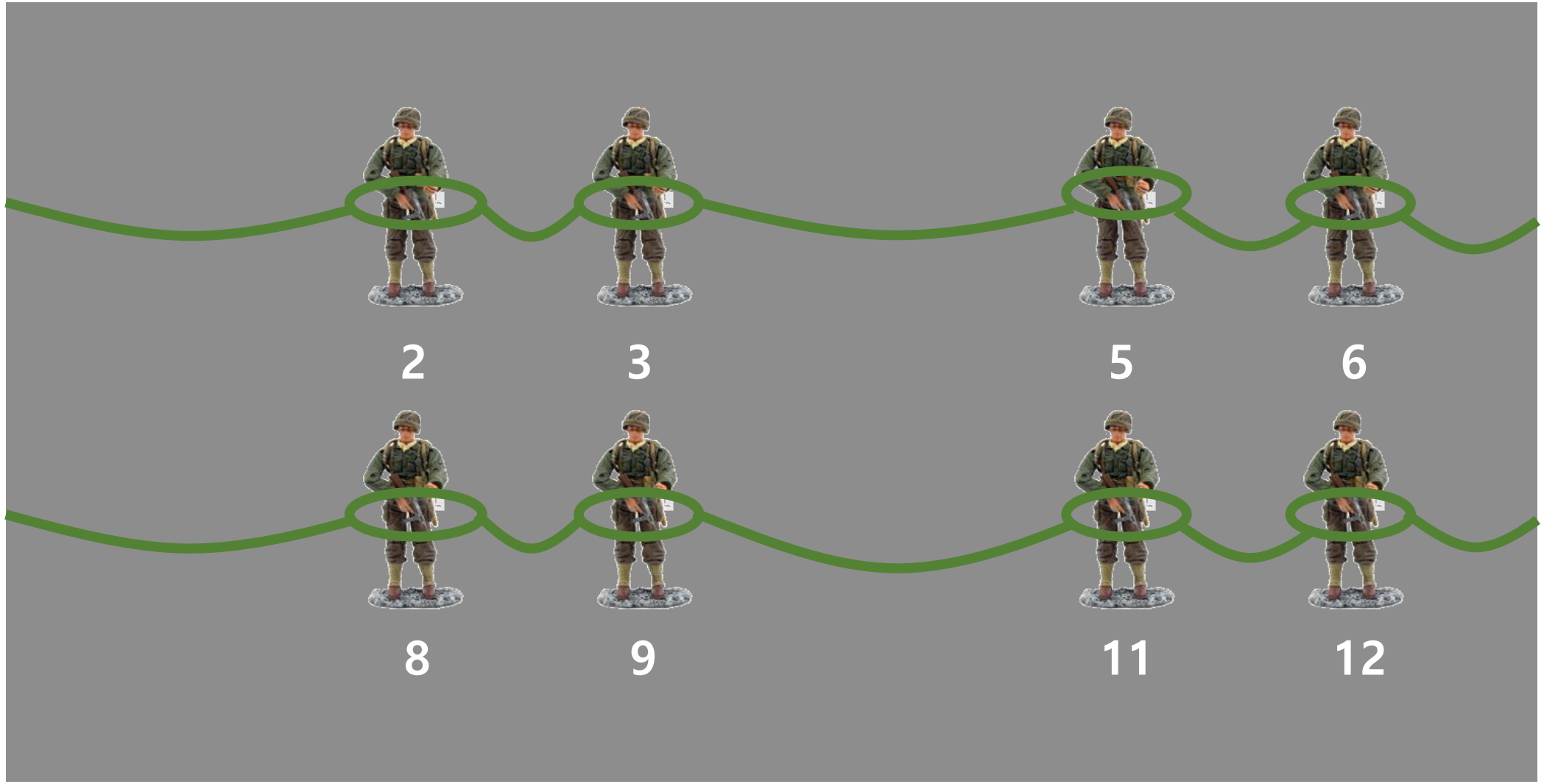
- **불필요한 연산 발생.**
  - 계속해서 배열 값을 확인해야 한다.
    (※ 만약, 위와 같이 하지 않는다면 몇 번째 병사인지 구하는 것에 문제가 생김.)

요세푸스 문제 (Josephus Problem)

- List를 이용하자!!!



요세푸스 문제 (Josephus Problem)



요세푸스 문제 (Josephus Problem)

- 노드를 생성하며, 각 노드에는 병사의 위치를 데이터로 저장한다.
- 첫 번째 병사를 삭제하고, 3만큼 이동한다.
- 병사를 삭제하고 3만큼 이동한다.
- 남은 병사 수가 2이면 각 병사의 데이터를 확인한다.

요세푸스 문제 (Josephus Problem)

- **List를 이용하자!!! - 장점**

- **불필요한 순회의 제거.**
  - 죽은 병사가 삭제 처리되므로, 순회해야 하는 남은 병사의 수 감소.

- **연산의 단순화.**
  - 이동 후 삭제라는 간단한 연산만으로 문제 해결 가능.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

삽입 정렬
(Insertion Sort)

삽입 정렬(Insertion Sort)

- 도서관 사서가 책을 정렬할 때, 일반적으로 활용되는 방식이 삽입 정렬이다.

- 자료 배열의 모든 원소들을 앞에서부터 차례대로 이미 정렬된 부분과 비교하여, 자신의 위치를 찾아냄으로써 정렬을 완성한다.

삽입 정렬(Insertion Sort)

정렬 과정

- 정렬할 자료를 두 개의 부분집합 S와 U로 가정
  - 부분집합 S : 정렬된 앞부분의 원소들
  - 부분집합 U : 아직 정렬되지 않은 나머지 원소들
- 정렬되지 않은 부분집합 U의 원소를 하나씩 꺼내서 이미 정렬 되어있는 부분집합 S의 마지막 원소부터 비교하면서 위치를 찾아 삽입한다.
- 삽입 정렬을 반복하면서 부분집합 S의 원소는 하나씩 늘리고 부분집합 U의 원소는 하나씩 감소하게 한다. 부분집합 U가 공집합이 되면 삽입정렬이 완성된다.

시간 복잡도

- $O(n^2)$

삽입 정렬(Insertion Sort)

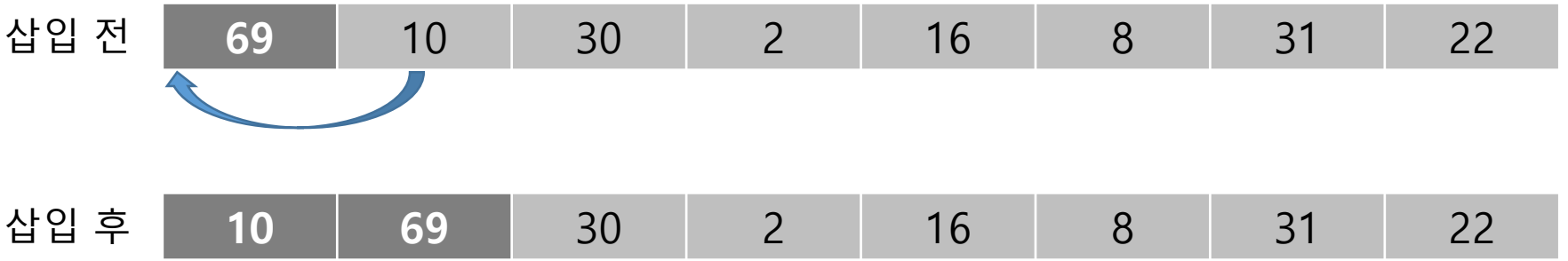
- **{69, 10, 30, 2, 16, 8, 31, 22}를 삽입 정렬하는 과정**
  - 초기 상태 : 첫 번째 원소는 정렬된 부분 집합 S로 생각하고 나머지 원소들은 정렬되지 않은 부분 집합 U로 생각한다.

69	10	30	2	16	8	31	22

S: 정렬된 원소 U: 미정렬된 원소

삽입 정렬(Insertion Sort)

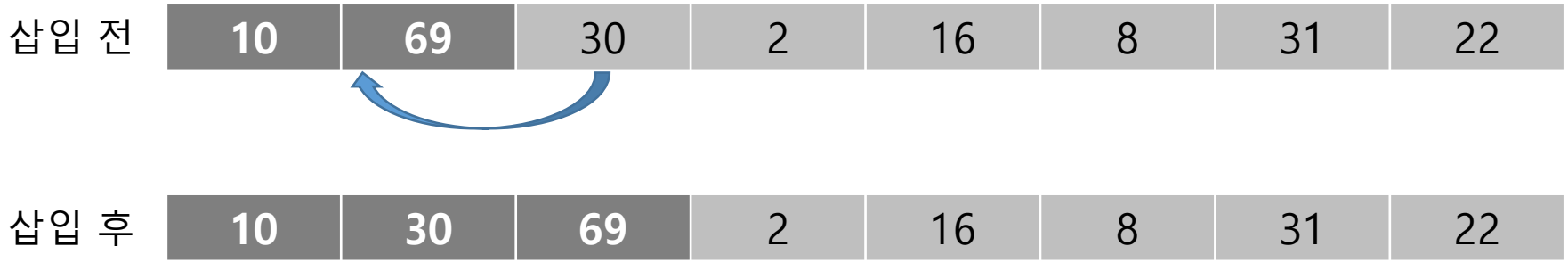
- U의 첫 번째 원소 10을 S의 마지막 원소 69와 비교할 때, 10<69이므로, 원소 10은 69의 앞자리에 위치하게 된다.
- 더 이상 비교할 S의 원소가 없으므로 찾은 위치에 원소 10을 삽입한다.



S: 정렬된 원소 U: 미정렬된 원소

삽입 정렬(Insertion Sort)

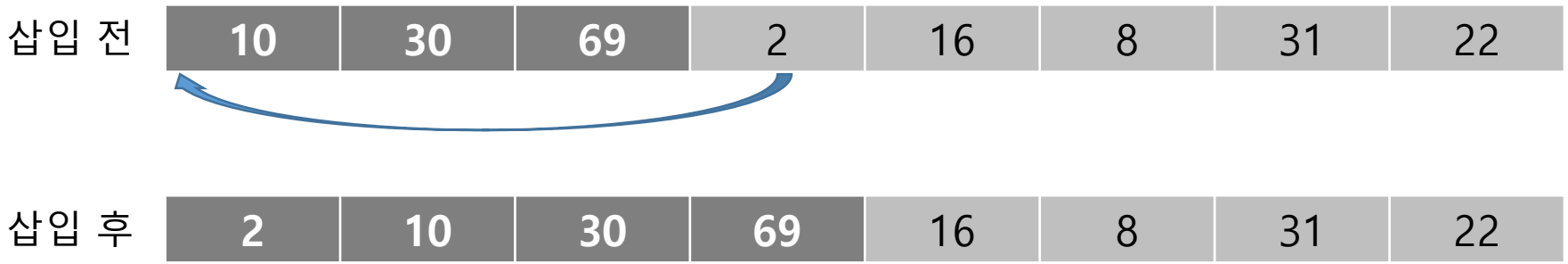
- U의 첫 번째 원소 30을 S의 마지막 원소 69와 비교할 때, 30<69이므로, 69의 앞자리 원소 10과 비교한다.
- 30 > 10 이므로 원소 10과 69 사이에 삽입한다.



S: 정렬된 원소 U: 미정렬된 원소

삽입 정렬(Insertion Sort)

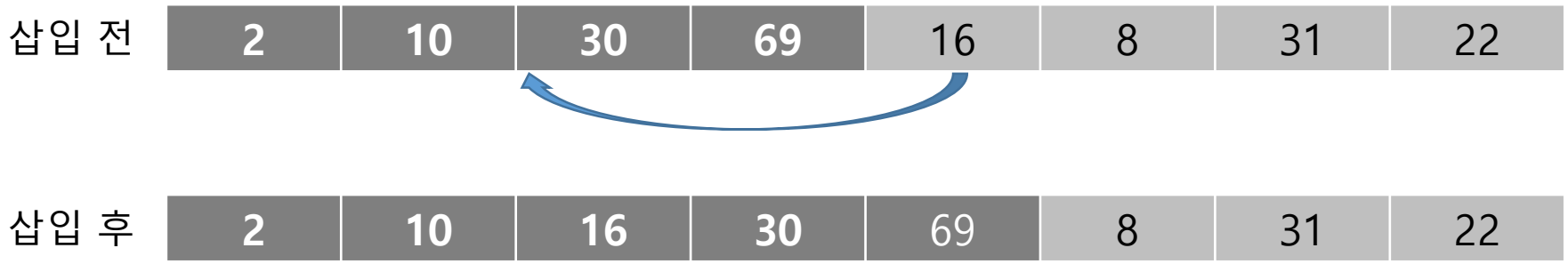
- U의 첫 번째 원소 2를 S의 마지막 원소 69와 비교할 때, 2<69이므로, 69의 앞자리 원소 30과 비교한다.
- 비교를 반복하여 최종적으로 가장 앞자리에 삽입한다.



S: 정렬된 원소 U: 미정렬된 원소

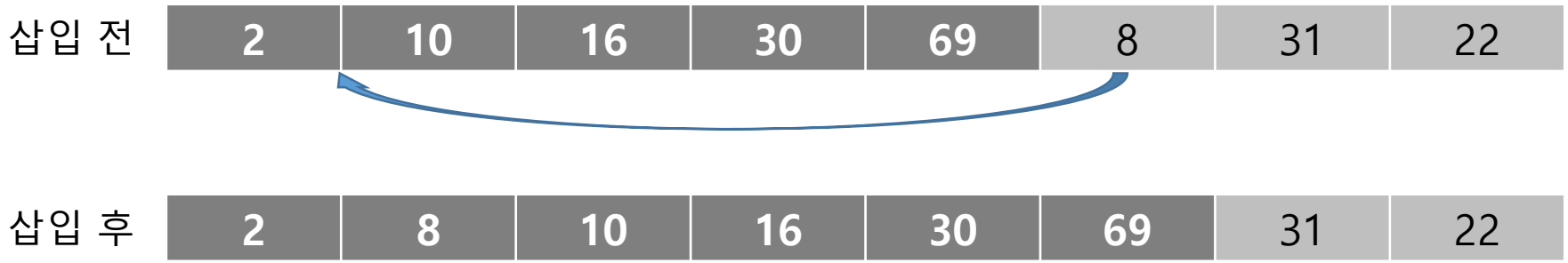
삽입 정렬(Insertion Sort)

- U의 첫 번째 원소 16을 S의 마지막 원소 69와 비교할 때, 16<69이므로, 69의 앞자리 원소 30과 비교한다.
- 비교를 반복하여 10과 30 사이에 삽입한다.



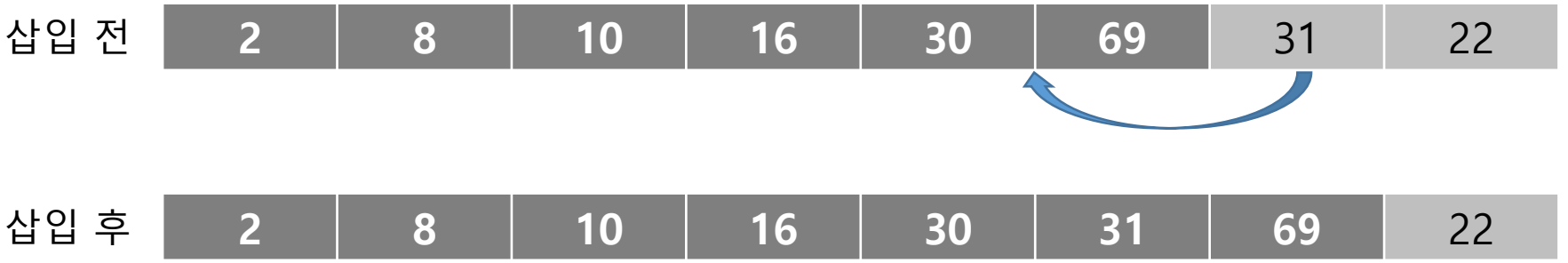
삽입 정렬(Insertion Sort)

- U의 첫 번째 원소 8을 S의 마지막 원소 69와 비교할 때, 8<69이므로, 69의 앞자리 원소 30과 비교한다.
- 비교를 반복하여 2와 10 사이에 삽입한다.



삽입 정렬(Insertion Sort)

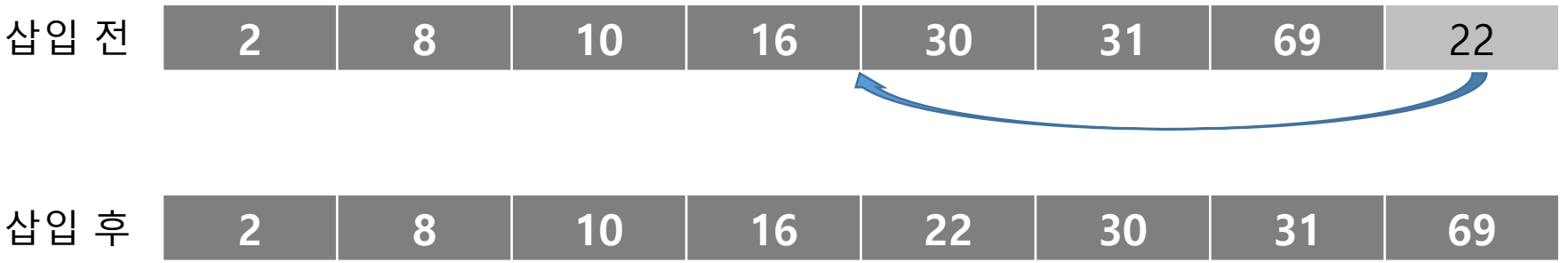
- U의 첫 번째 원소 31을 S의 마지막 원소 69와 비교할 때, 31<69이므로, 69의 앞자리 원소 30과 비교한다.
- 비교를 반복하여 30와 69 사이에 삽입한다.



S: 정렬된 원소 U: 미정렬된 원소

삽입 정렬(Insertion Sort)

- U의 첫 번째 원소 22을 S의 마지막 원소 69와 비교할 때, 22<69이므로, 69의 앞자리 원소 31과 비교한다.
- 비교를 반복하여 16와 30 사이에 삽입한다.



S: 정렬된 원소 U: 미정렬된 원소

삽입 정렬(Insertion Sort)

- 학습한 정렬 알고리즘의 특성을 다른 정렬들과 비교해보자.

알고리즘	평균 수행시간	최악 수행시간	알고리즘 기법	비고
버블 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	코딩이 가장 손쉽다.
선택 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	교환의 회수가 버블, 삽입정렬보다 작다.
카운팅 정렬	O(n+k)	O(n+k)	비교환 방식	n이 비교적 작을 때만 가능하다.
삽입 정렬	$O(n^2)$	$O(n^2)$	비교와 교환	n의 개수가 작을 때 효과적이다.
병합 정렬	O(n log n)	O(n log n)	분할 정복	연결리스트의 경우 가장 효율적인 방식
퀵 정렬	O(n log n)	$O(n^2)$	분할 정복	최악의 경우 $O(n^2)$ 이지만, 평균적으로는 가장 빠르다.

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

다음 방송에서 만나요!

삼성 청년 SW 아카데미