알튜비튜
투 포인터

두 개의 포인터로 배열을 빠르게 탐색하는 알고리즘입니다.
코딩 테스트에선 주로 효율성을 보는 문제에 활용됩니다.

이와 더불어 투 포인터와 함께 자주 활용되는 누적 합, 슬라이딩 윈도우에 대해서도 알아봅니다.

도전 문제 1

2473번: 세 용액 - Gold 3

문제

- 이전 수업 시간에 다루었던 두 용액 문제의 응용 버전
- 용액의 특성값의 합이 가장 0에 가까운 세 용액을 구하는 문제

제한 사항

- 전체 용액의 수 n: 3 ≤ n ≤ 1000
- 용액의 특성값: -1,000,000,000 이상 1,000,000,000 이하

예제 입력

```
5
-2 6 -97 -6 98
```

예제 출력

```
-97 -2 98
```

두 용액 문제를 떠올려 볼까요…?

#알튜비튜

/<> 2470번 : 두 용액 - Gold 5

문제

- 두 개의 서로 다른 용액을 혼합해, 합이 0에 가까운 용액을 만들어라

제한 사항

- 용액의 수 N은 2 <= N <= 100,000
- 용액의 특성값 k는 -1e9 <= k <= 1e9 (-10억 ~ 10억)

예제 입력

```
5
-2 4 -99 -1 98
```

예제 출력

```
-99 98
```

두 용액 문제를 떠올려 볼까요…?

Left + Right = 76

Ans = 76

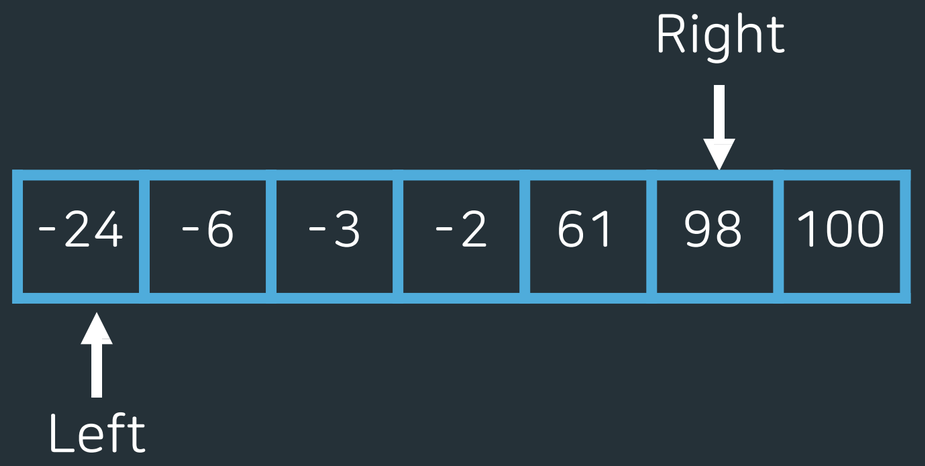
두 용액 문제를 떠올려 볼까요…?

Left + Right = 76

Ans = 76

0보다 크니까 숫자를 줄이자!

두 용액 문제를 떠올려 볼까요…?



Left + Right = 74

Ans = 74

두 용액 문제를 떠올려 볼까요…?

Left + Right = 37

Ans = 37

두 용액 문제를 떠올려 볼까요…?

Left + Right = -26

Ans = -26

두 용액 문제를 떠올려 볼까요?

#알튜비튜

- 두 용액 문제에서는 투 포인터를 두 용액으로 대응시켰었죠?
- 그렇다면 세 용액에서는…?
  => 두 용액에서와 마찬가지로 투 포인터 사용
  => 가장 주의해야 할 것은 중복이 되는 연산이 없어야 한다는 것!

  => 반드시 포함되는 용액을 정해서 투 포인터 연산 범위를 한정시켜줍시다

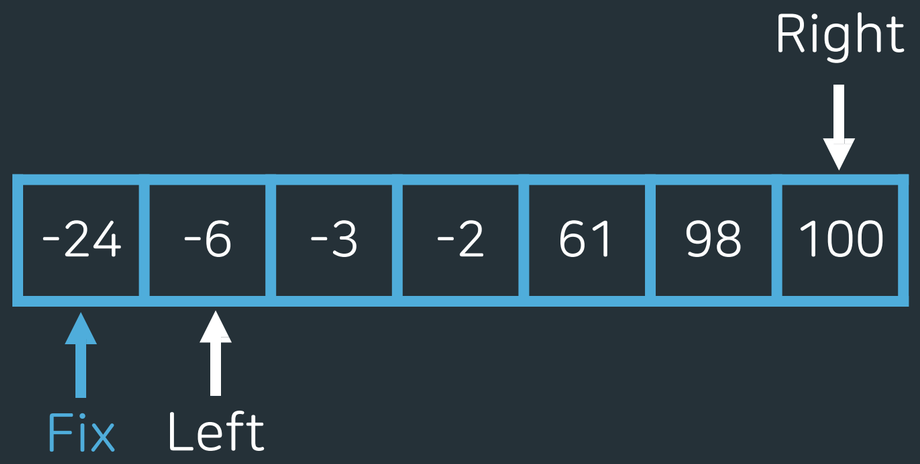
먼저 모든 용액을 정렬해줍시다

2023-1 SW학부 원스탑 튜터 알튜비튜

투 포인터? 아니 쓰리 포인터!

투 포인터? 아니 쓰리 포인터!

투 포인터? 아니 쓰리 포인터!

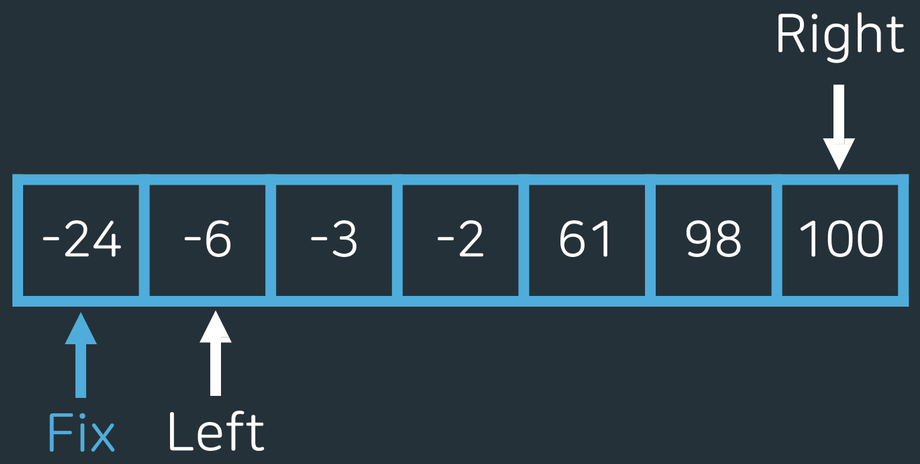


Fix 포인터로 반드시 포함할 용액 지정
Fix보다 왼쪽에 있는 용액에 대해서는 탐색 X

Fix를 0번째 인덱스에서부터 오른쪽으로 이동시켜가며 세 용액의 합 계산

2023-1 SW학부 원스탑 튜터 알튜비튜

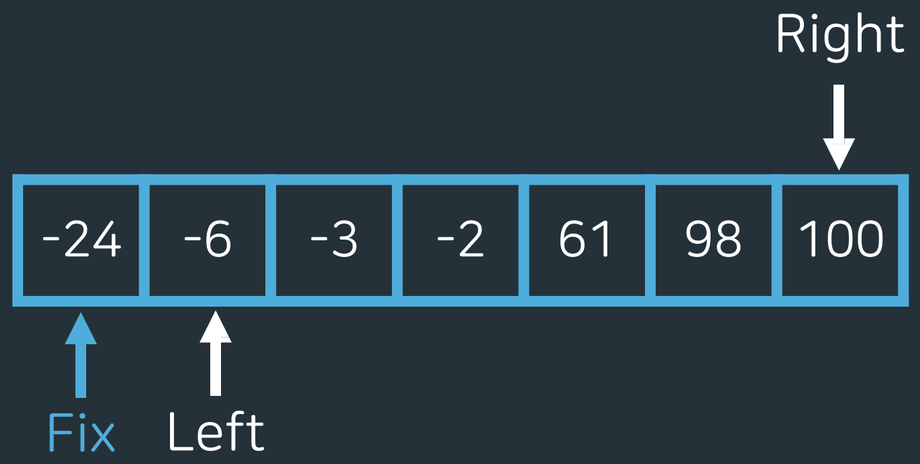
투 포인터? 아니 쓰리 포인터!



ans = INF

2023-1 SW학부 원스탑 튜터 알튜비튜

투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = 70
ans = 70

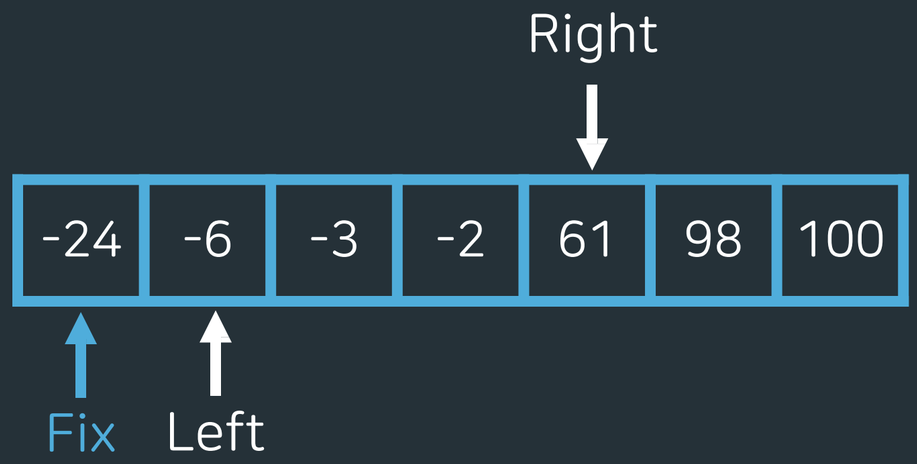
2023-1 SW학부 원스탑 튜터
알튜비튜

투 포인터? 아니 쓰리 포인터!

Fix + Left + Right = 68
ans = 68

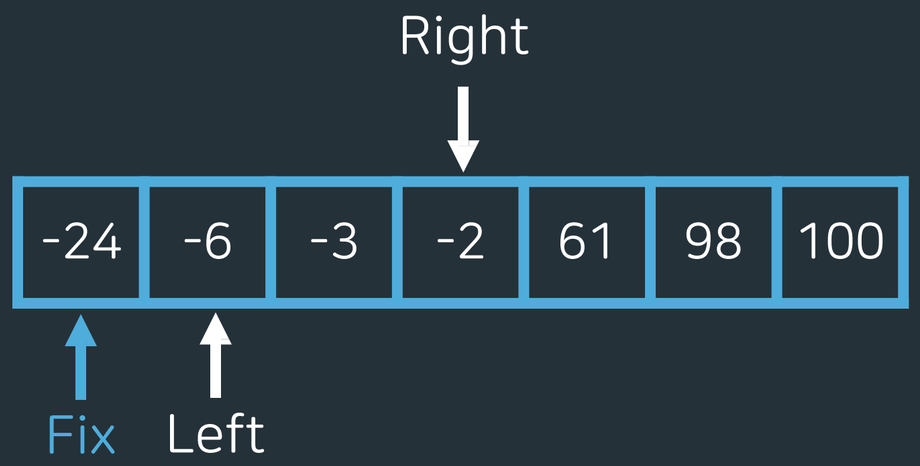
2023-1 SW학부 원스탑 튜터
알튜비튜

투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = 31
ans = 31

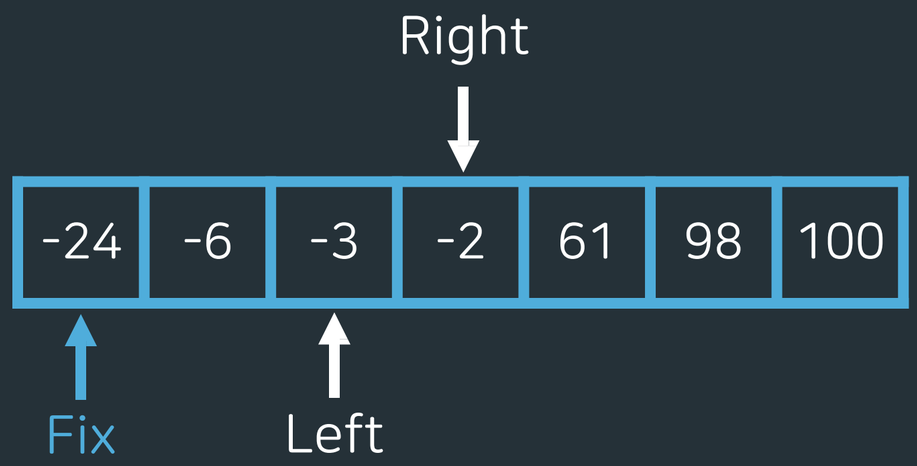
투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = -32
ans = 31

2023-1 SW학부 원스탑 튜터 알튜비튜

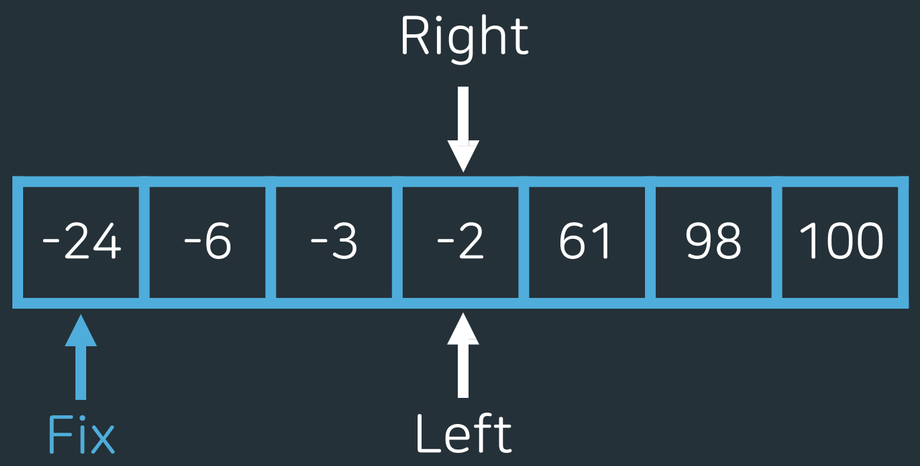
투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = -29
ans = -29

2023-1 SW학부 원스탑 튜터
알튜비튜

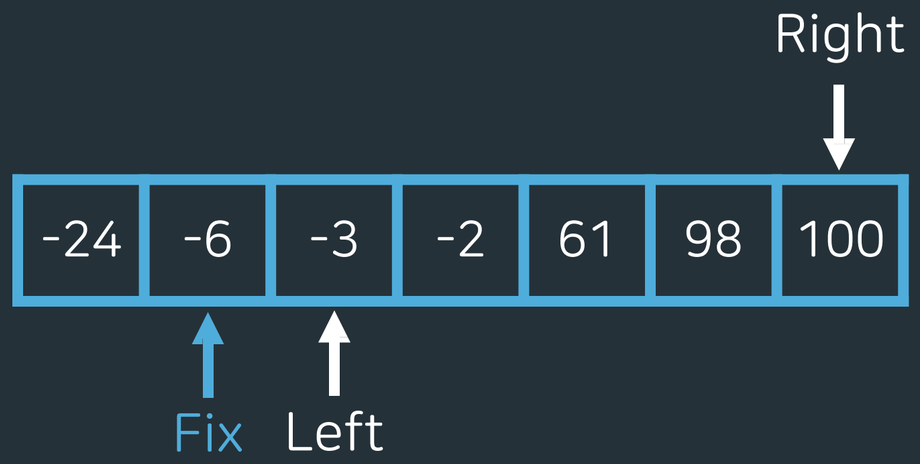
투 포인터? 아니 쓰리 포인터!



서로 다른 세 용액이어야 하므로 break
Fix를 오른쪽으로 이동

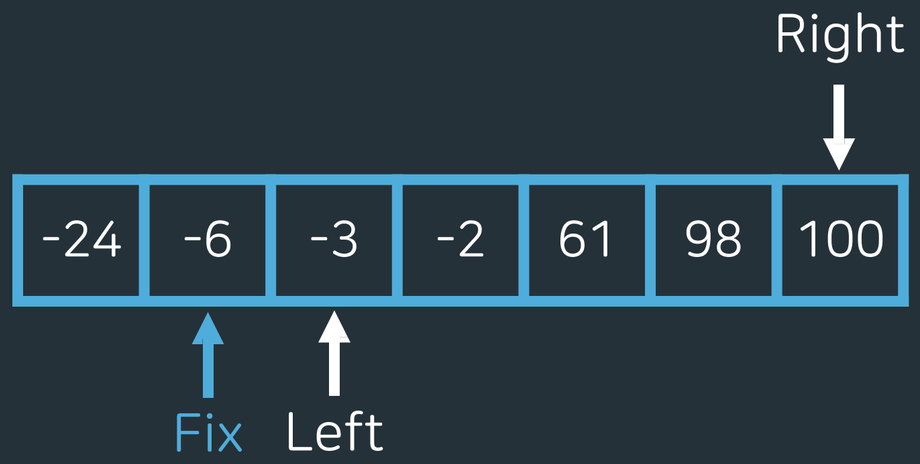
2023-1 SW학부 원스탑 튜터 알튜비튜

투 포인터? 아니 쓰리 포인터!



2023-1 SW학부 원스탑 튜터
알튜비튜

투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = 91

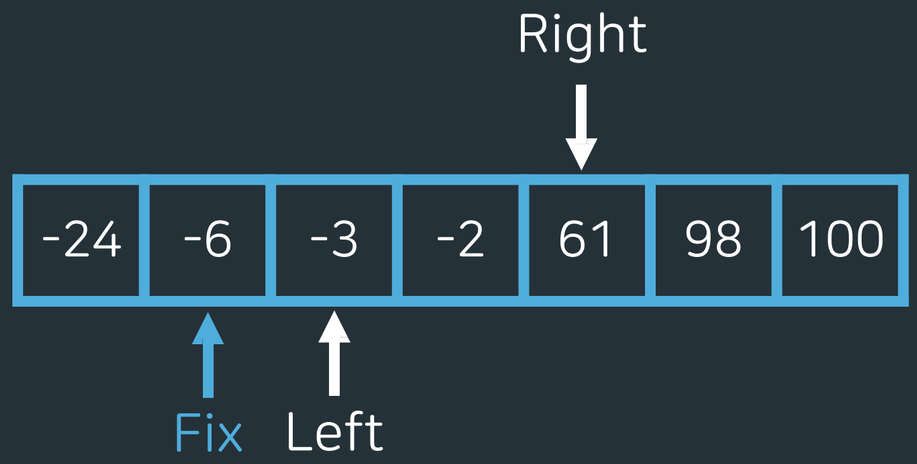
ans = -29

투 포인터? 아니 쓰리 포인터!

Fix + Left + Right = 89

ans = -29

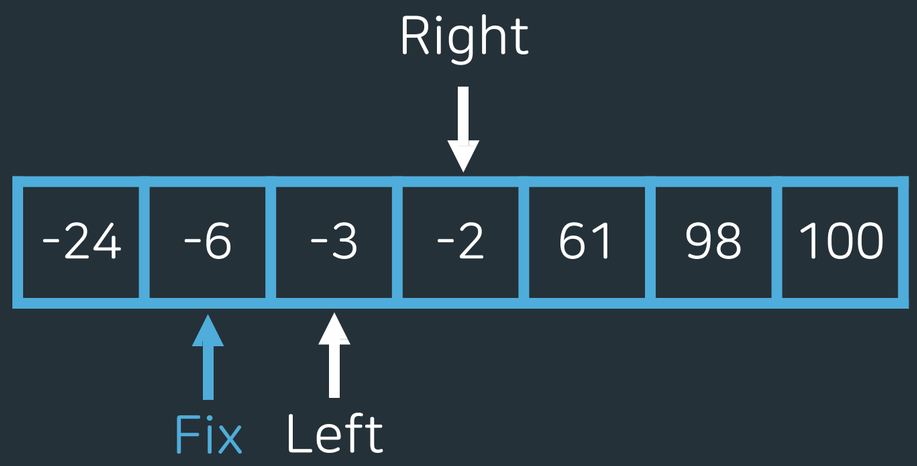
투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = 52

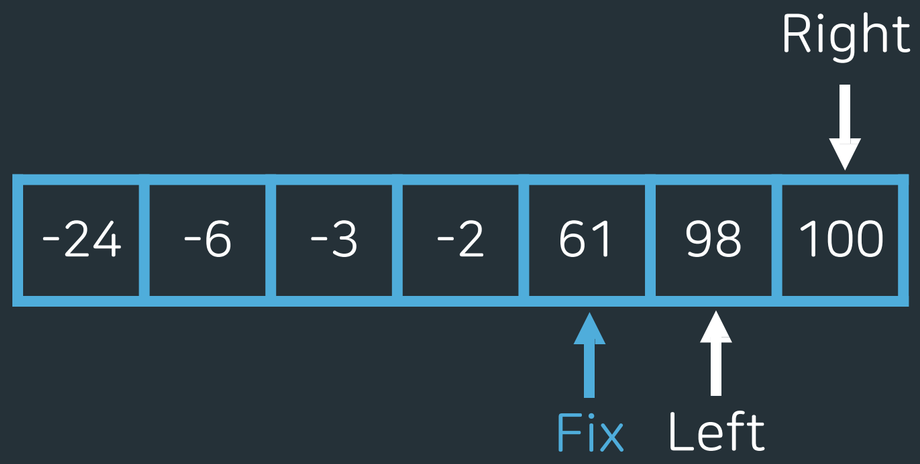
ans = -29

투 포인터? 아니 쓰리 포인터!



Fix + Left + Right = -11
ans = -11

투 포인터? 아니 쓰리 포인터!



Fix가 n-3이 되면 탐색 종료

도전 문제 2

#알튜비튜

13422번: 도둑 - Gold 4

문제

- n개의 집이 원형으로 배치된 마을에서 연속한 m개의 집을 도둑질
- 훔친 돈이 k보다 작을 때만 도둑질 성공
- 위 조건을 만족시키면서 훔칠 수 있는 방법의 가짓수를 구하는 문제
- 숨어 있는 독특한 코너케이스를 찾아내는 게 문제의 핵심!

제한 사항

- 집의 개수 n: $1 \le N \le 100,000$
- 돈을 훔칠 연속된 집의 개수 m: $1 \le M \le N$
- 자동 방법 장치가 작동하는 최소 돈의 양 k: $1 \le K \le 1,000,000,000$

도전 문제 2

예제 입력

```
2
8 3 15
3 4 7 5 6 4 2 9
2 1 5
4 5
```

예제 출력

```
3
1
```

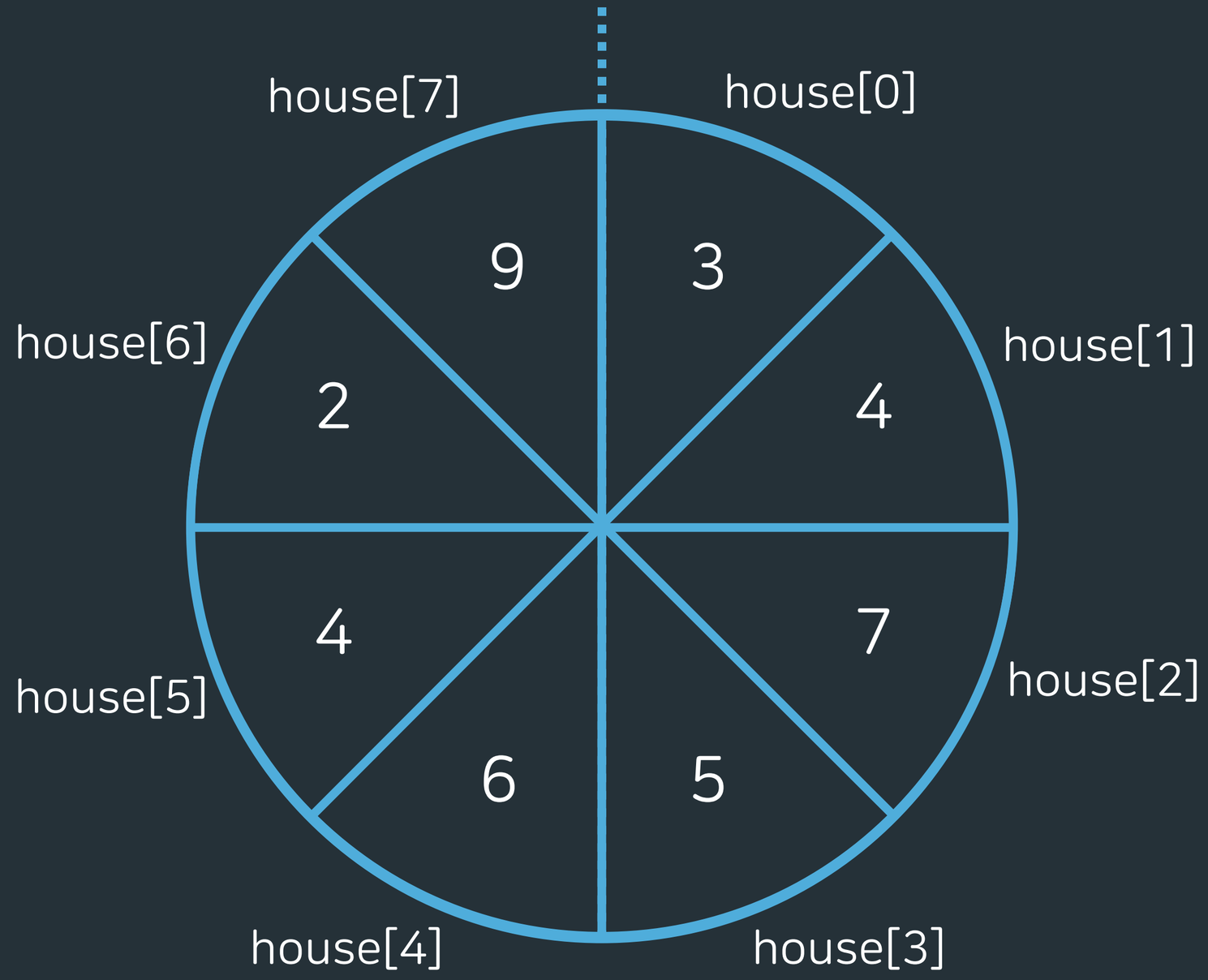
도전 문제 2

#알튜비튜

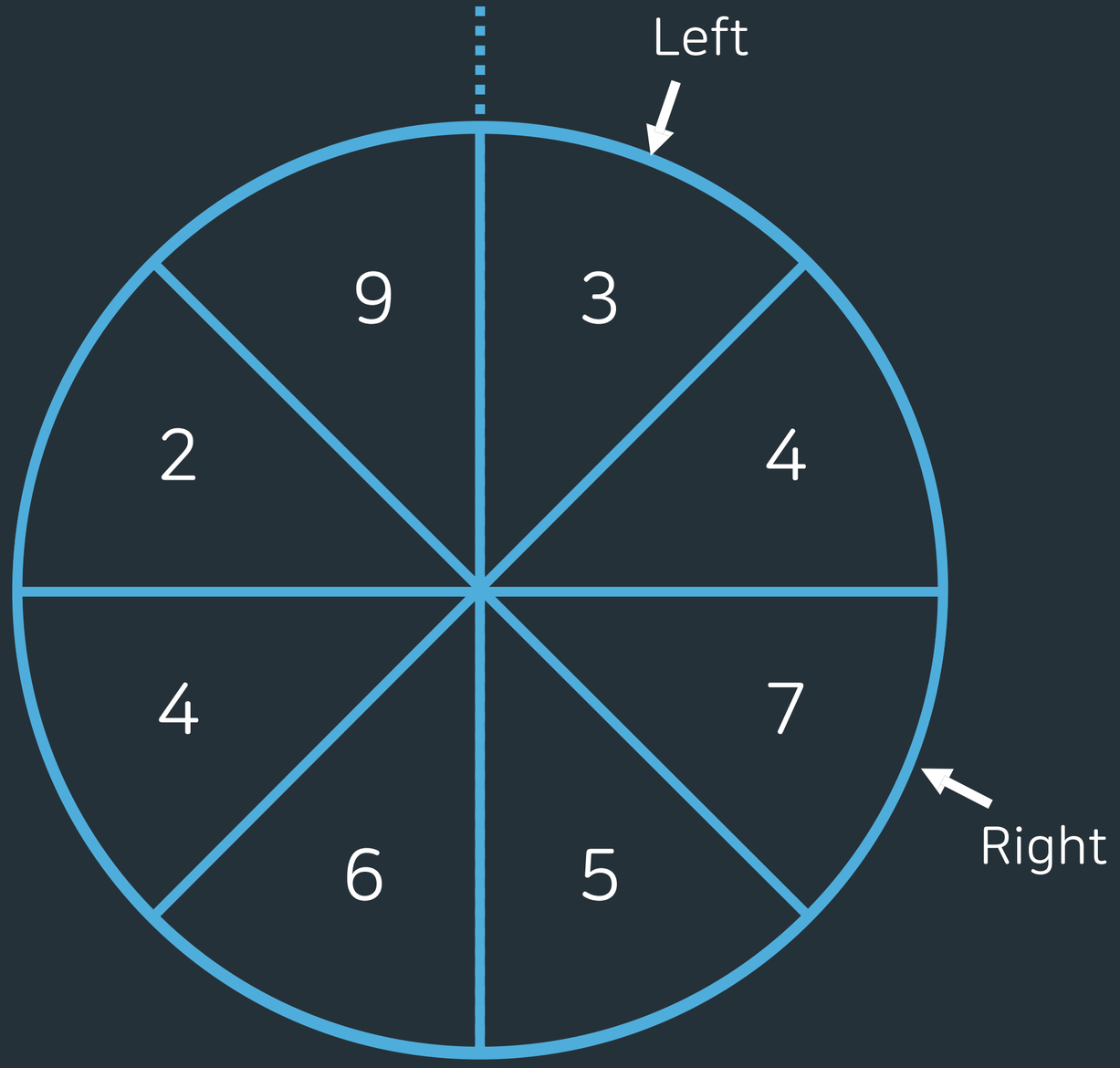
- 항상 m개의 집을 연속해서 도둑질해야 하는 상황
- 윈도우의 크기가 정해져 있을 때 사용할 수 있었던 알고리즘이 있었죠?
- 마을이 원형으로 생겨서 누적 합은 적용하기 까다로울 것 같아요..

  => 슬라이딩 윈도우를 사용합시다!

돈을 훔쳐봅시다


house[7]
house[0]
9
3
house[6]
house[1]
2
4
4
7
house[5]
house[2]
6
5
house[4]
house[3]

돈을 훔쳐봅시다



돈을 훔쳐봅시다

A[0] + A[1] + A[2]
=14 < k

∴ ans = 1

돈을 훔쳐봅시다

A[1] + A[2] + A[3]
=16 > k

∴ ans = 1

돈을 훔쳐봅시다

A[2] + A[3] + A[4]
=18 > k

∴ ans = 1

돈을 훔쳐봅시다

A[7] + A[0] + A[1]
=16 > k

∴ ans = 3

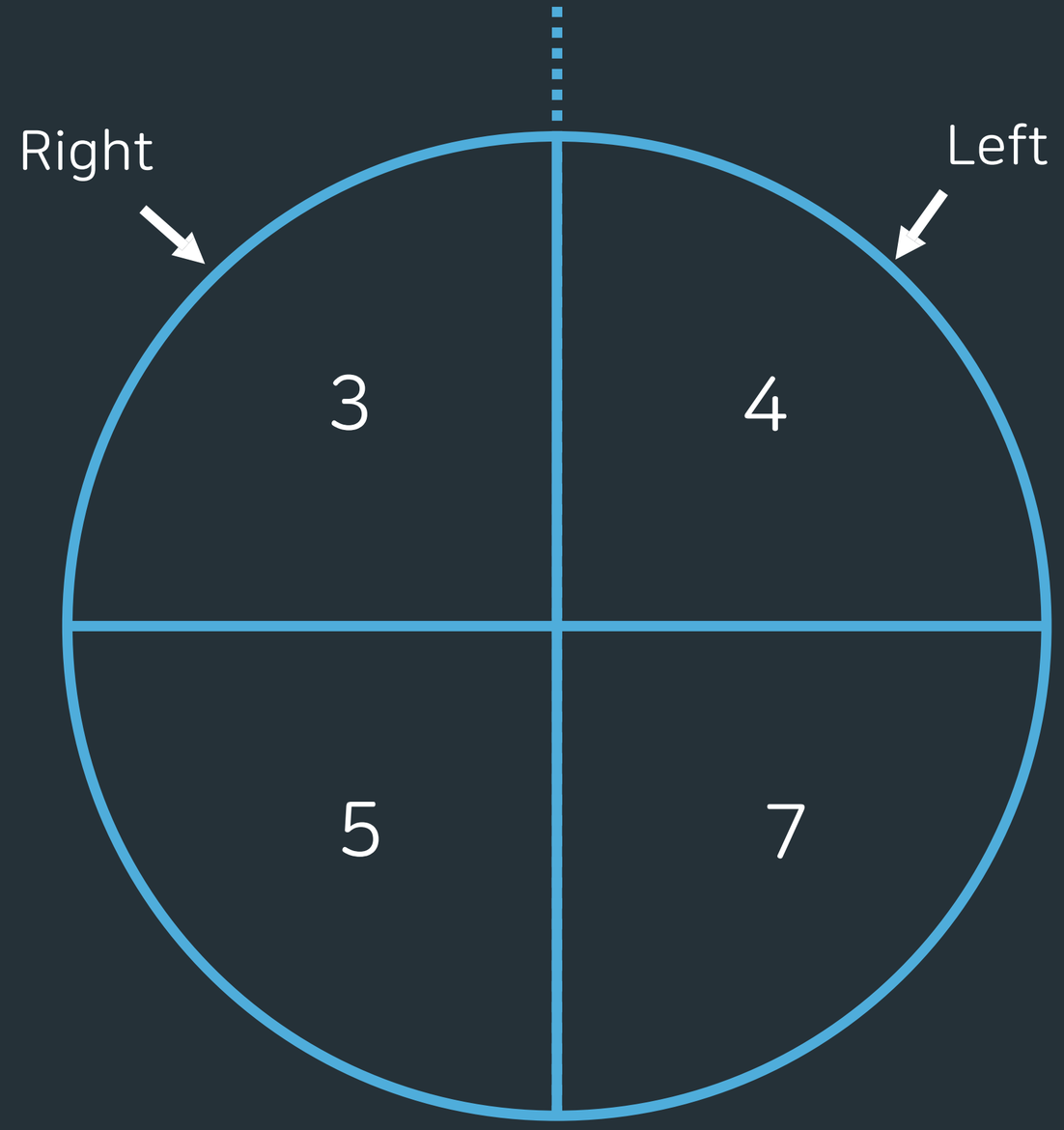
돈을 훔쳐봅시다

Left = 8 > n

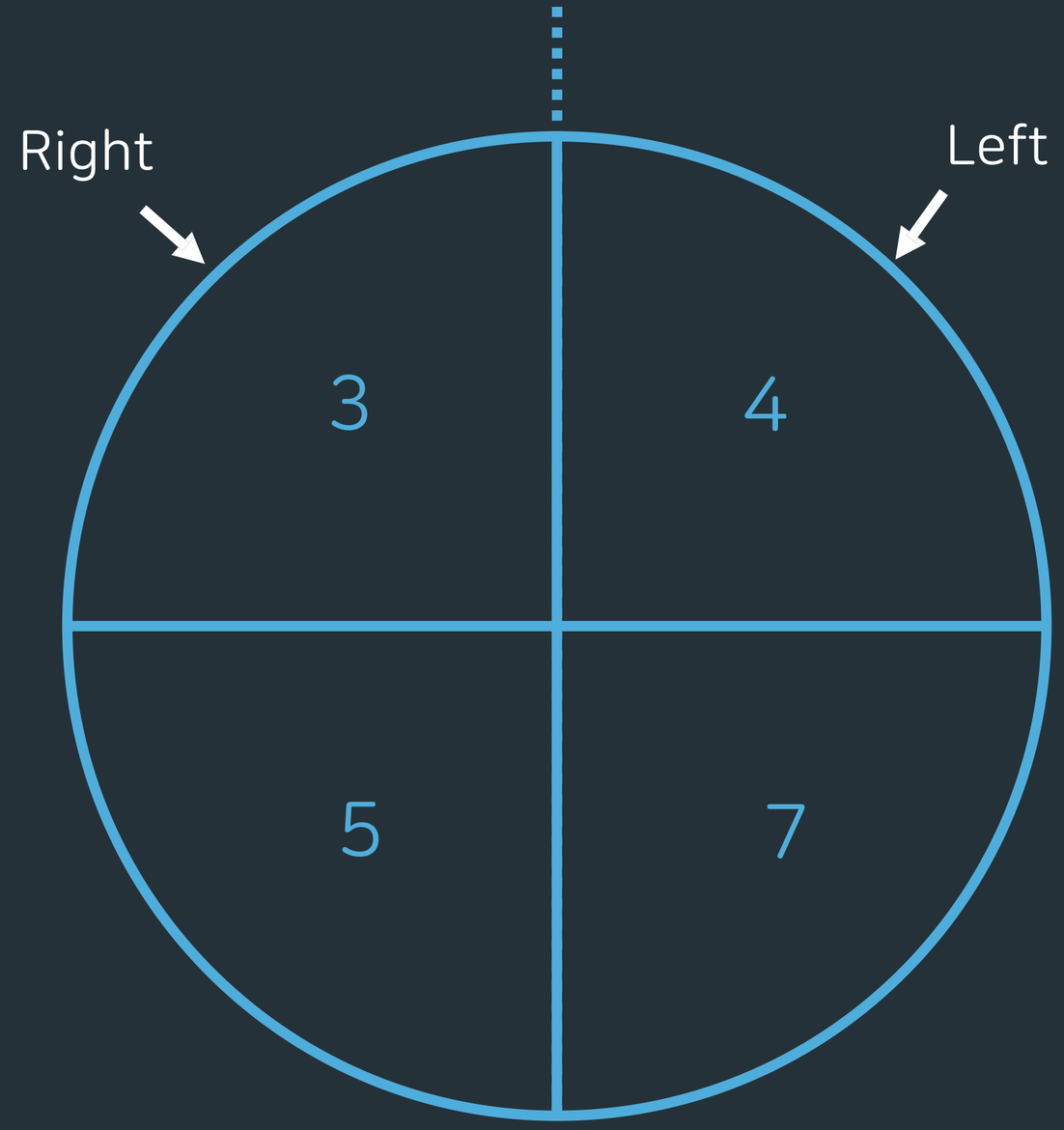
Left가 n을 넘어가면
연산 중단

n==m이라면…?

- n = 4
- m = 4
- k = 20
- 정답: 1



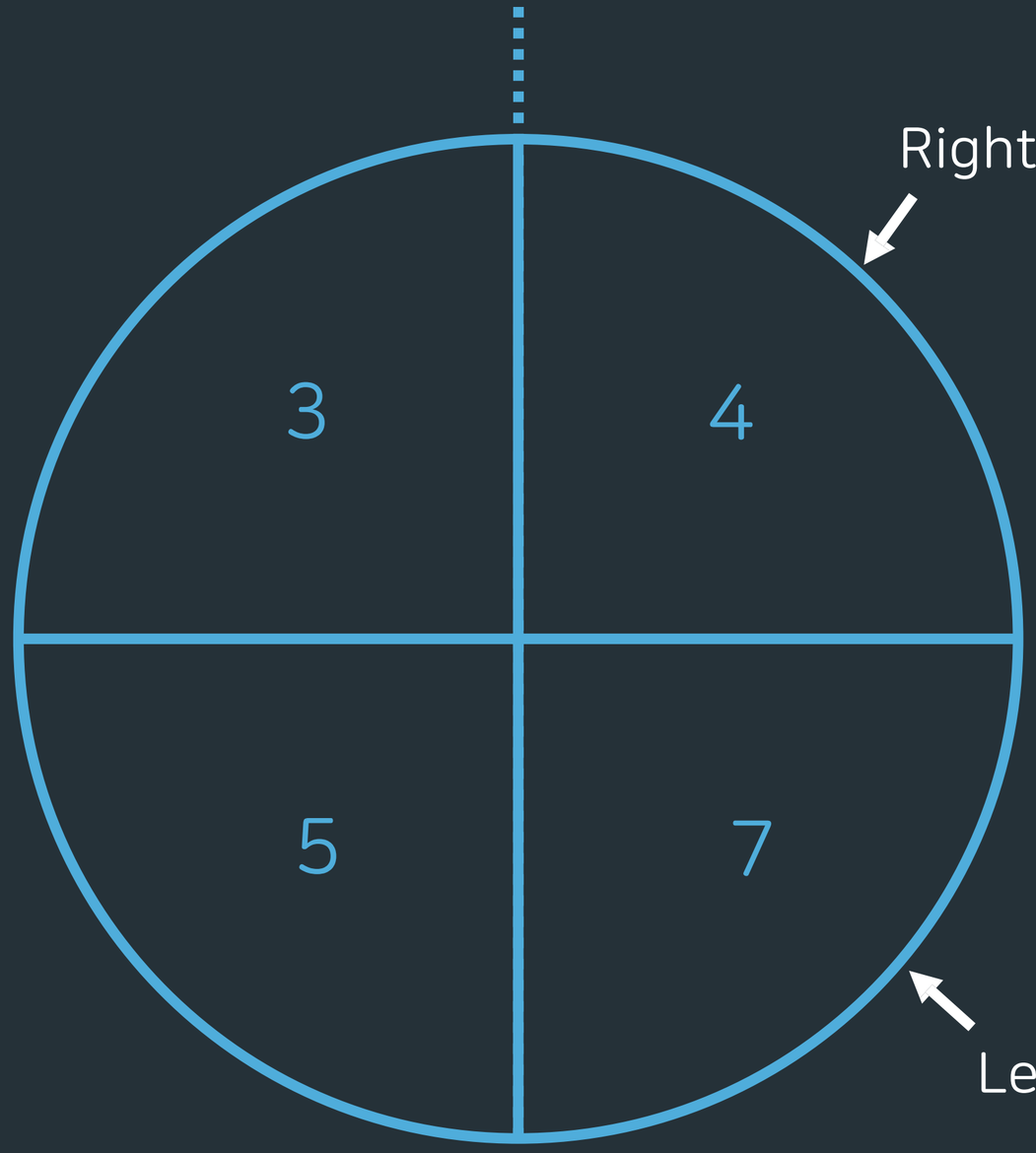
n==m이라면…?



A[0] + A[1] + A[2] + A[3]
= 19 < k

∴ ans = 1

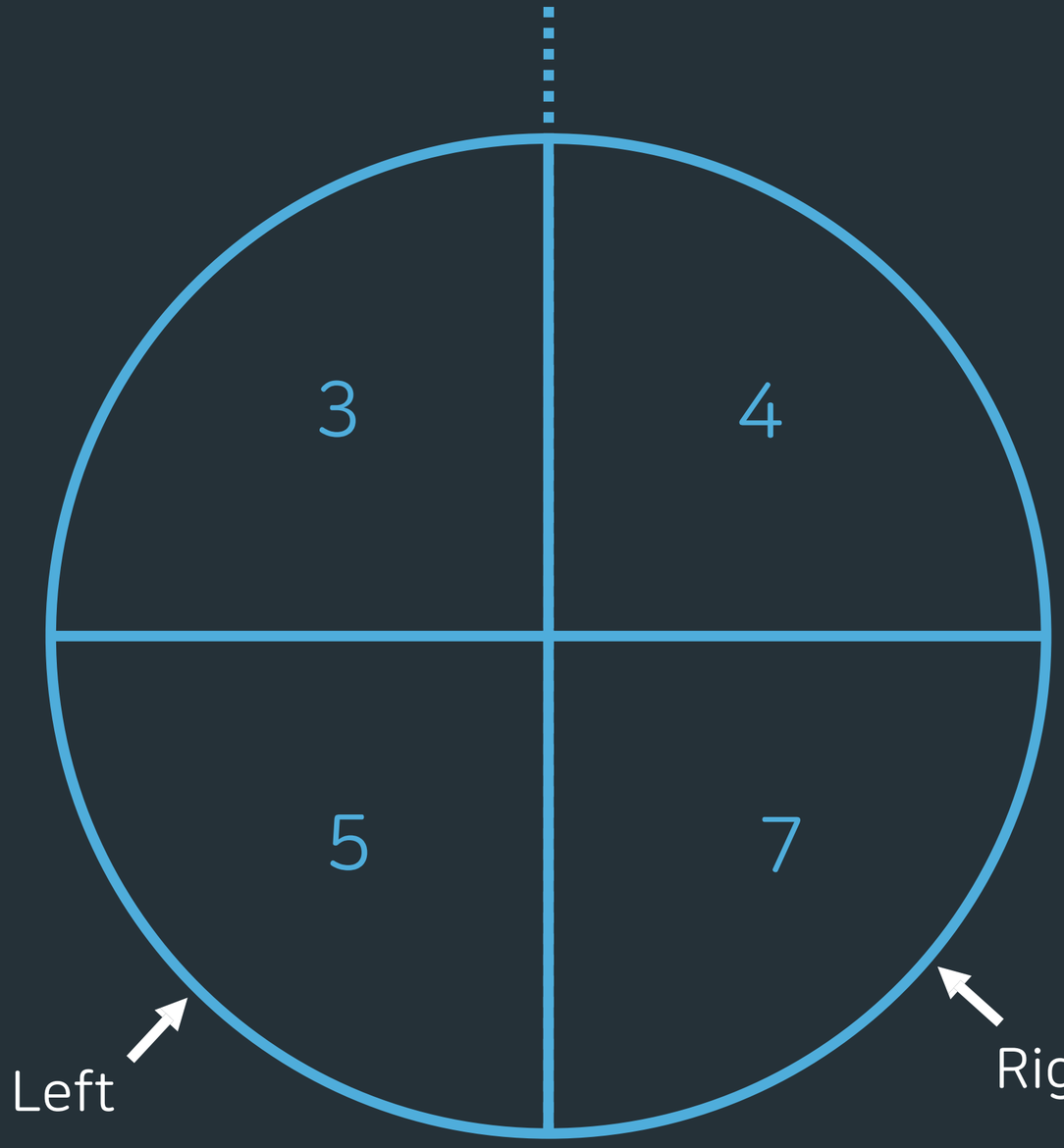
n==m이라면…?



A[1] + A[2] + A[3] + A[0]
= 19 < k

∴ ans = 2

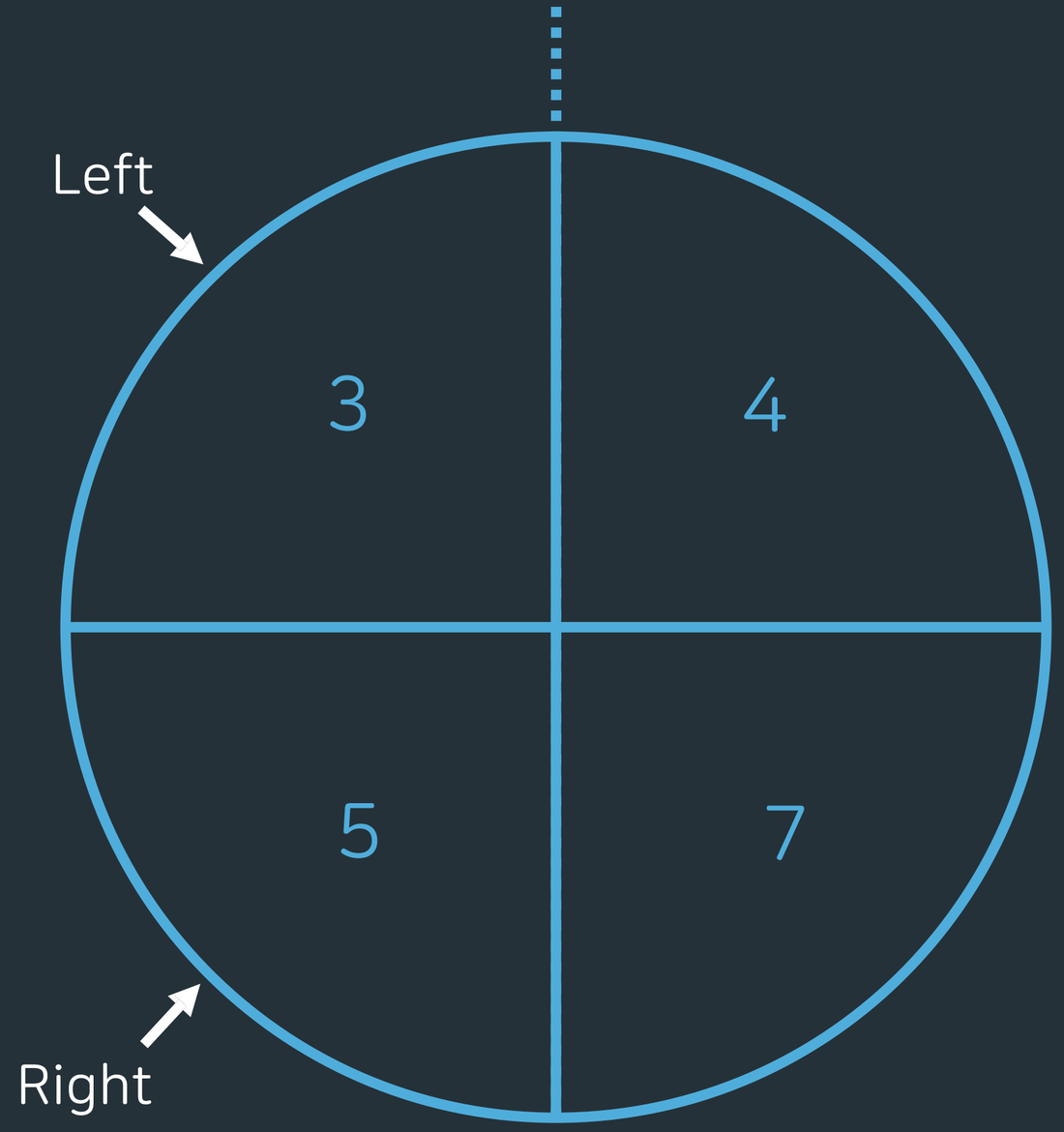
n==m이라면…?



A[2] + A[3] + A[0] + A[1]
= 19 < k

∴ ans = 3

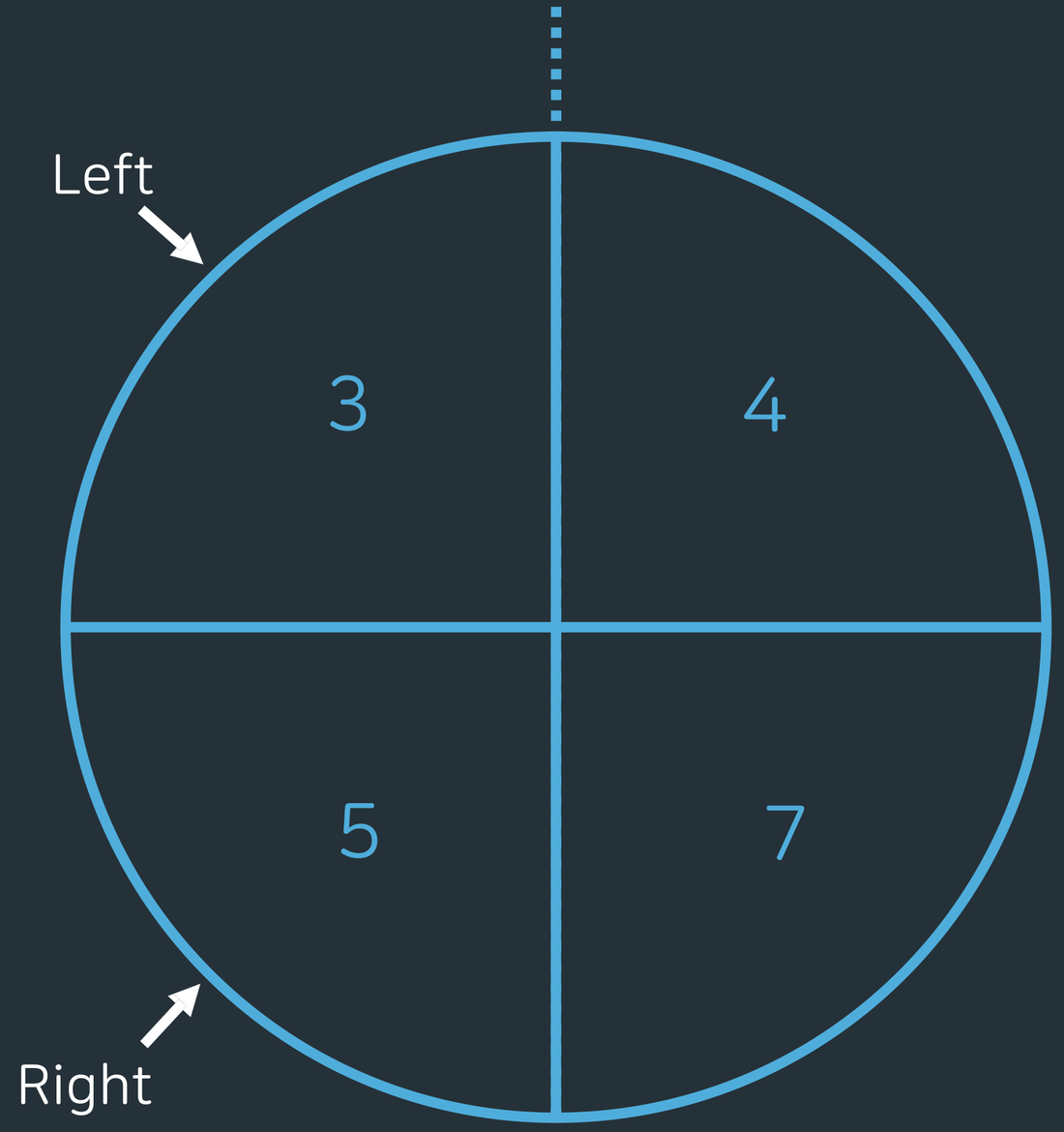
n==m이라면…?



A[3] + A[0] + A[1] + A[2]
= 19 < k

∴ ans = 4

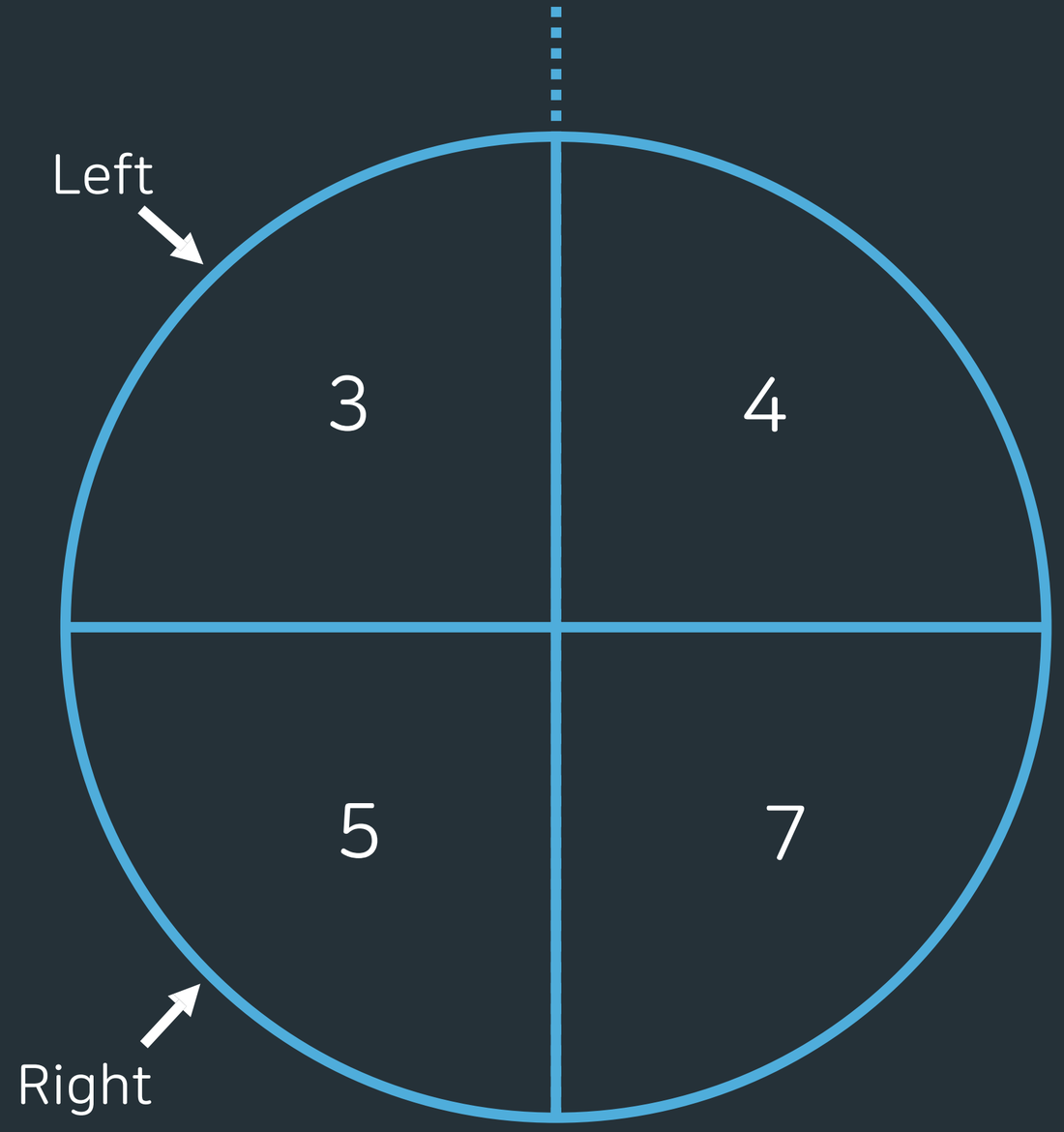
n==m이라면…?



A[3] + A[0] + A[1] + A[2]
= 19 < k

∴ ans = 4
정답 = 1

n==m이라면…?



A[3] + A[0] + A[1] + A[2]
= 19 < k

∴ ans = 4
정답 = 1

⇒ n == m인 경우에는
탐색을 반복해도 계속
동일한 연산이 이뤄지므로
반드시 분류할 것!!

14503번 : 로봇 청소기 - Gold 5

문제

로봇 청소기와 방의 상태가 주어졌을 때, 청소하는 영역의 개수를 구하는 프로그램 만들기

1. 현재 칸이 아직 청소되지 않은 경우, 현재 칸 청소
2. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 없는 경우,
    1. 바라보는 방향을 유지한 채로 한 칸 후진할 수 있다면 한 칸 후진하고 1번으로 돌아감
    2. 바라보는 방향의 뒤쪽 칸이 벽이라 후진할 수 없다면 작동 멈춤
3. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 있는 경우,
    1. 반시계 방향 90º 회전
    2. 바라보는 방향을 기준으로 앞쪽 칸이 청소되지 않은 빈 칸인 경우 한 칸 전진하고 1번으로 돌아감

필수 문제

#알튜비튜

제한 사항

- 방의 크기 N과 M의 범위 3 <= N, M <= 50

예제 입력

```
3 3
1 1 0
1 1 1
1 0 1
1 1 1
```

예제 출력

```
1
```

접근법

#알튜비튜

1. 현재 칸이 아직 청소되지 않은 경우, 현재 칸 청소
2. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 없는 경우,
    1. 바라보는 방향을 유지한 채로 한 칸 후진할 수 있다면 한 칸 후진하고 1번으로 돌아감
    2. **바라보는 방향의 뒤쪽 칸이 벽이라 후진할 수 없다면 작동 멈춤**
3. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 있는 경우,
    1. 반시계 방향 90^{o} 회전
    2. 바라보는 방향을 기준으로 앞쪽 칸이 청소되지 않은 빈 칸인 경우 한 칸 전진하고 1번으로 돌아감

Hint

1. 2-2번을 만족하기 전까지 로봇 청소기는 작동을 멈추지 않아요! 무한 반복문 안에서 작동하다가 2-2번을 만족할 때 break로 반복문을 빠져나와 볼까요?
2. 문제 내용 그대로 구현하는 게 중요해요. 조건과 방향에 유의하여 구현합시다!

의사코드

1. 현재 칸이 아직 청소되지 않은 경우, 현재 칸 청소
2. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 없는 경우,
    1. 바라보는 방향을 유지한 채로 한 칸 후진할 수 있다면 한 칸 후진하고 1번으로 돌아감
    2. 바라보는 방향의 뒤쪽 칸이 벽이라 후진할 수 없다면 작동 멈춤
3. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 있는 경우,
    1. 반시계 방향 90º 회전
    2. 바라보는 방향을 기준으로 앞쪽 칸이 청소되지 않은 빈 칸인 경우 한 칸 전진하고 1번으로 돌아감

```
무한반복문 {
        if(아직 청소되지 않은 경우)  // 1번 작업
                현재 칸 청소

        bool 빈칸 = false
        주변4칸 탐색 시작 { // 반시계 방향으로 회전
                if(청소되지 않은 빈칸 발견) {
                        빈칸 = true
                        한 칸 전진
                        break
                }
        }

        if(빈칸 == true) // 주변 4칸 중 빈 칸 있었음
                continue; // 1번 작업으로

        if(후진할 수 있는 경우)
                한 칸 전진

        else  // 후진할 수 없는 경우
                break // 작동 종료
}
```

의사코드

#알튜비튜

1. 현재 칸이 아직 청소되지 않은 경우, 현재 칸 청소
2. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 없는 경우,
    1. 바라보는 방향을 유지한 채로 한 칸 후진할 수 있다면 한 칸 후진하고 1번으로 돌아감
    2. 바라보는 방향의 뒤쪽 칸이 벽이라 후진할 수 없다면 작동 멈춤
3. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 있는 경우,
    1. 반시계 방향 90º 회전
    2. 바라보는 방향을 기준으로 앞쪽 칸이 청소되지 않은 빈 칸인 경우 한 칸 전진하고 1번으로 돌아감

```
무한반복문 {
        if(아직 청소되지 않은 경우)  // 1번 작업
                현재 칸 청소

        bool 빈칸 = false
        주변4칸 탐색 시작 { // 반시계 방향으로 회전
                if(청소되지 않은 빈칸 발견) {
                        빈칸 = true
                        한 칸 전진
                        break
                }
        }

        if(빈칸 == true) // 주변 4칸 중 빈 칸 있었음
                continue; // 1번 작업으로

        if(후진할 수 있는 경우)
                한 칸 전진

        else  // 후진할 수 없는 경우
                break // 작동 종료
}
```

의사코드

#알튜비튜

1. 현재 칸이 아직 청소되지 않은 경우, 현재 칸 청소
2. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 없는 경우,
    1. 바라보는 방향을 유지한 채로 한 칸 후진할 수 있다면 한 칸 후진하고 1번으로 돌아감
    2. 바라보는 방향의 뒤쪽 칸이 벽이라 후진할 수 없다면 작동 멈춤
3. 현재 칸의 주변 4칸 중 청소되지 않은 빈 칸이 있는 경우,
    1. 반시계 방향 90º 회전
    2. 바라보는 방향을 기준으로 앞쪽 칸이 청소되지 않은 빈 칸인 경우 한 칸 전진하고 1번으로 돌아감

```
무한반복문 {
        if(아직 청소되지 않은 경우)  // 1번 작업
                현재 칸 청소

        bool 빈칸 = false
        주변4칸 탐색 시작 { // 반시계 방향으로 회전
                if(청소되지 않은 빈칸 발견) {
                        빈칸 = true
                        한 칸 전진
                        break
                }
        }

        if(빈칸 == true) // 주변 4칸 중 빈 칸 있었음
                continue; // 1번 작업으로

        if(후진할 수 있는 경우)
                한 칸 전진

        else  // 후진할 수 없는 경우
                break // 작동 종료
}
```

DFS로 풀이할 수는 없나요?

#알튜비튜

Hint

DFS는 최대한 깊게 탐색했다가
더 이상 탐색할 수 없는 경우, 가까운 갈림길도 돌아와 다시 탐색을 진행하는 방법이에요.
하지만 로봇 청소기는 2-2번 조건을 만족하면 즉시 작동을 멈추네요.

가까운 갈림길로 되돌아와 탐색하지 않도록 구현하면 DFS로도 풀이할 수 있어요!
DFS 풀이도 올려놓았으니 확인해보세요 ☺

마무리

#알튜비튜

추가로 풀어보면 좋은 문제!

- 6159번 : 코스튬 파티 – Silver 5
- 2531번 : 회전 초밥 – Silver 1
- 16472번 : 고냥이 – Gold 4
- 1484번 : 다이어트 – Gold 4