알튜비튜
분할 정복

한 번에 해결할 수 없는 문제를 작게 쪼개어 풀어나가는 분할 정복에 대해 배워봅시다.

이런 문제가 있다고 해봅시다.

"자연수 N, K가 주어질 때, N^k를 구하는 방법?"

이런 문제가 있다고 해봅시다.

"자연수 N, K가 주어질 때, N^k를 구하는 방법?"

N을 K번 곱하면 되지 않나? 🤔

일단 곱해봅시다!

#알튜비튜

```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 2, k = 10;
    int ans = 1;

    while (k--)
        ans *= n;

    cout << ans;
}
```

이건 좀 오래 걸리지 않을까요?


```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 2, k = 1000000000;
    long long ans = 1;

    while (k--)
        ans *= n;

    cout << ans;
}
```

Divide and Conquer

- 한 번에 해결할 수 없는 문제를 작은 문제로 분할하여 해결하는 알고리즘
- 큰 문제 -> 작은 문제 => Top-Down 접근!
- 주로 재귀 함수로 구현
- 분할 방법에 따라 시간 복잡도는 천차만별
- 입력 범위 N이 큰 편
- 크게 3단계로 이루어짐
  1. Divide : 문제 분할
  2. Conquer : 쪼개진 작은 문제 해결
  3. Combine : 해결된 작은 문제들을 다시 합침

아까 그 문제를 다시 풀면?


```cpp
#include <iostream>
#include <cmath>

using namespace std;

long long divide(int n, int k) {
    if (k == 1) //Conquer
        return n;
    //Divide + Combine
    if (k % 2 == 0)
        return pow(divide(n, k / 2), 2);
    return n * divide(n, k - 1);
}

int main() {
    int n = 2, k = 1000000000;

    cout << divide(n, k);
}
```

K가 1이라면, $N^k == N$ (=기저 조건)
K가 짝수라면, $N^k == N^{(K/2)} * N^{(K/2)}$
K가 홀수라면, $N^k == N * N^{(K-1)}$

시간 복잡도 비교

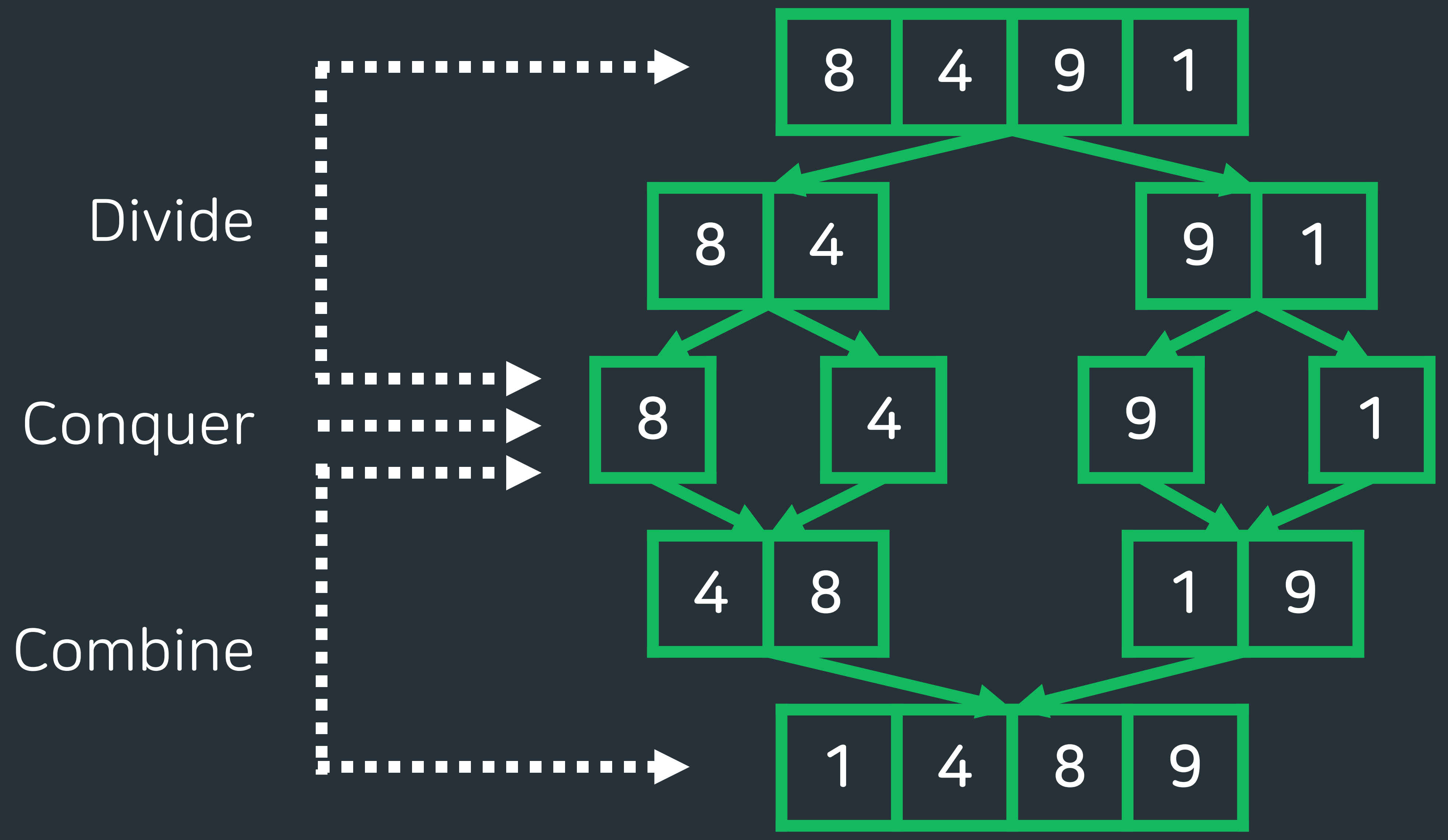

```cpp
#include <iostream>

using namespace std;

int main() {
    int n = 2, k = 1000000000;
    long long ans = 1;

    while (k--)
        ans *= n;

    cout << ans;
}
```

O(n)

```cpp
#include <iostream>
#include <cmath>

using namespace std;

long long divide(int n, int k) {
    if (k == 1) //Conquer
        return n;
    //Divide + Combine
    if (k % 2 == 0)
        return pow(divide(n, k / 2), 2);
    return n * divide(n, k - 1);
}

int main() {
    int n = 2, k = 1000000000;

    cout << divide(n, k);
}
```

O(logn)

뭔가 낯설지 않은 이 기분



합병 정렬은 대표적인 분할 정복 문제!

분할 방법에 따라 시간 복잡도가 다르다?

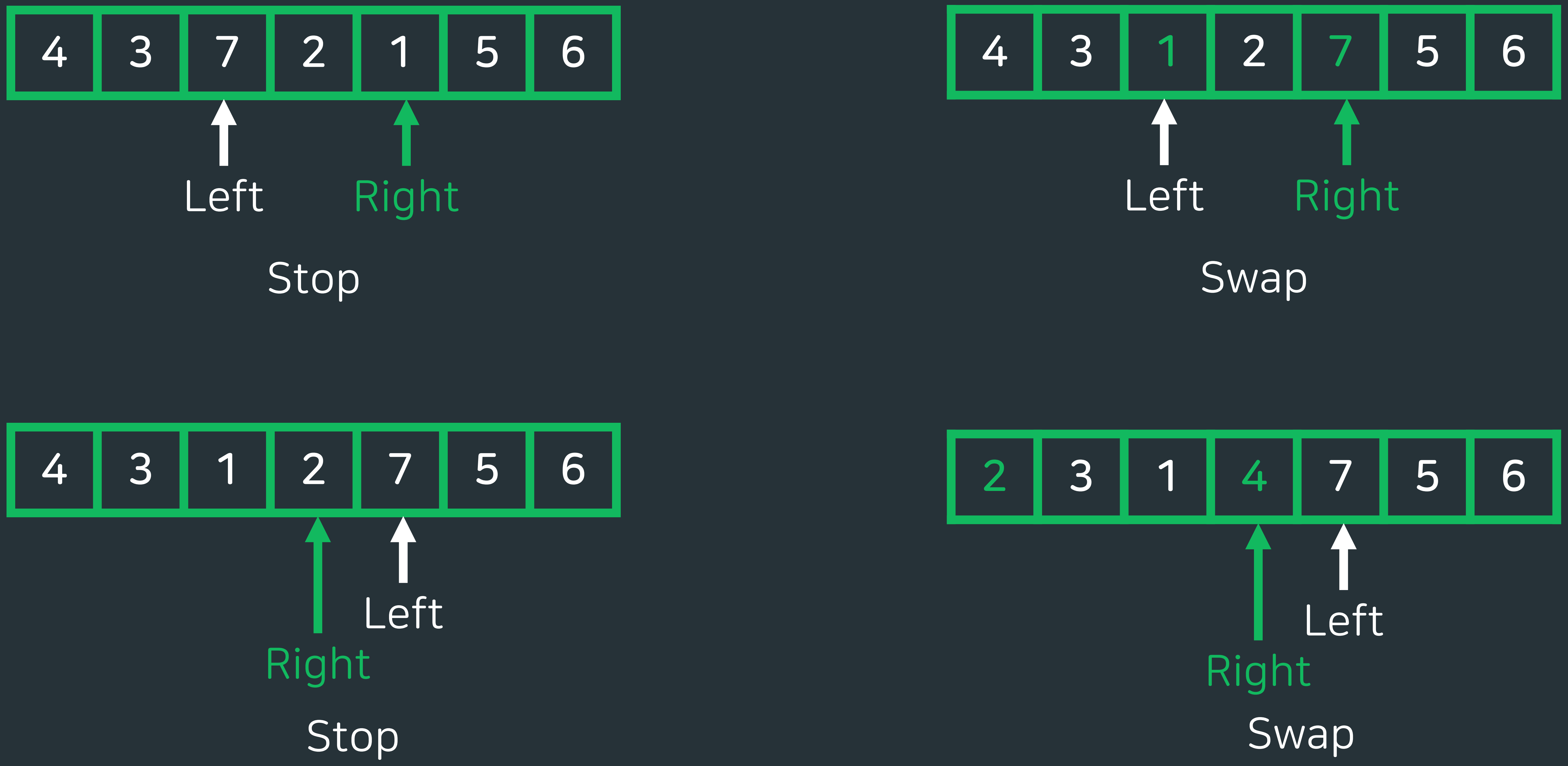
Quick sort

- 피봇을 기준으로 배열을 둘로 나눈 뒤 정렬
- 시간 복잡도는 일반적으로 O(nlogn)
- 피봇의 선택 방법에 따라 최악의 경우 $O(n^2)$의 시간 복잡도가 될 수 있음
- 정렬 과정
  1. 배열에서 원소 하나(피봇)을 선택
  2. 배열의 왼쪽엔 피봇보다 작은 원소, 오른쪽엔 피봇보다 큰 원소 배치
  3. 왼쪽, 오른쪽 배열에 대해 1~2 과정 반복
  4. 배열의 크기가 0 또는 1이 되면 종료

간단한 퀵 정렬 설명

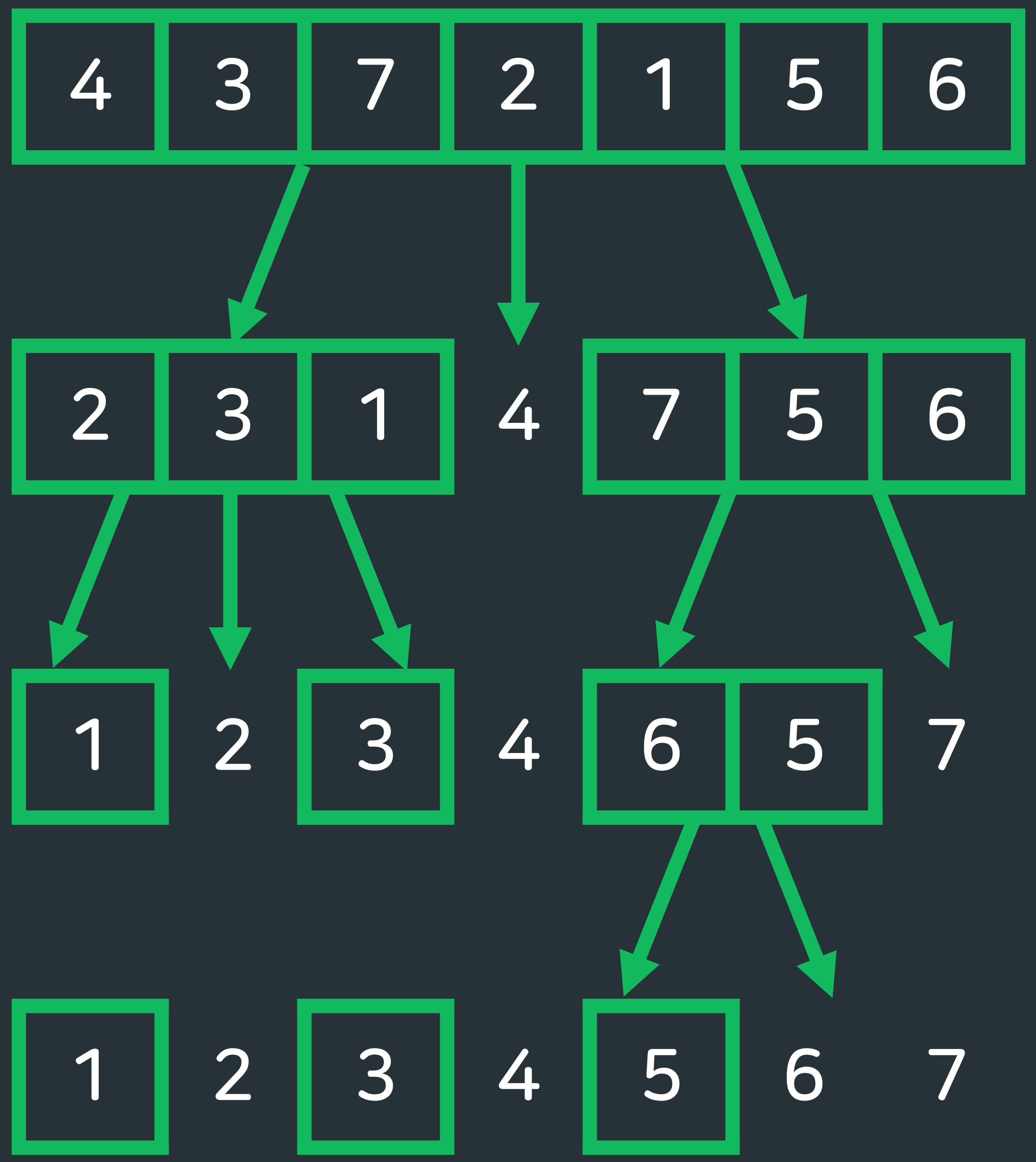
1. Pivot의 다음 위치를 Left, 범위의 마지막 위치를 Right로 설정
2. Left가 가리키는 값이 Pivot보다 클 때까지 오른쪽으로 이동
3. Right가 가리키는 값이 Pivot보다 작을 때까지 왼쪽으로 이동
4. 두 포인터가 모두 멈추면 각 포인터가 가리키는 원소를 Swap
5. Left < Right일 동안 2~4 반복
6. Right가 가리키는 원소와 Pivot이 가리키는 원소를 Swap

간단한 퀵 정렬 설명



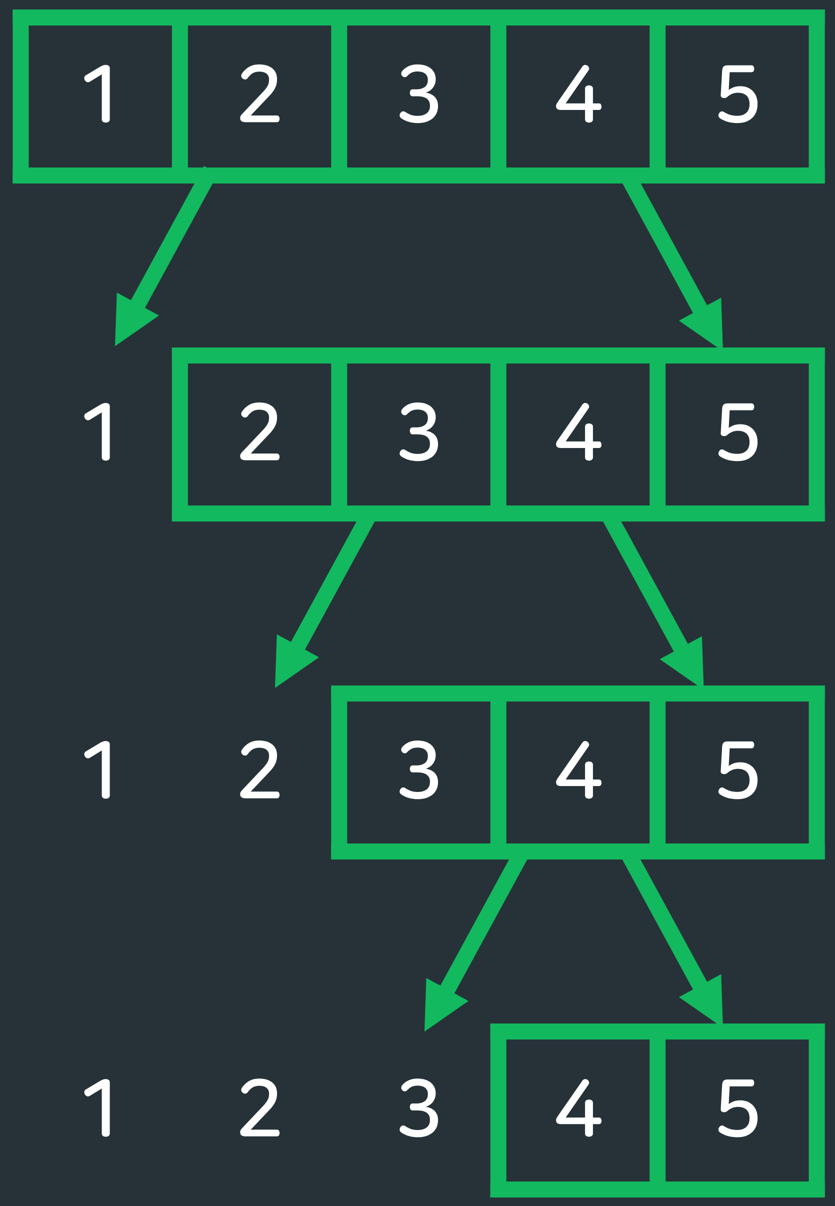
4 3 7 2 1 5 6
Left
Right
Stop
4 3 1 2 7 5 6
Left
Right
Swap
4 3 1 2 7 5 6
Left
Right
Stop
2 3 1 4 7 5 6
Left
Right
Swap

일반적인 경우



O(nlogn)

최악의 경우



$O(n^2)$

1629번 : 곱셈 – Silver 1

문제

- 자연수 A를 B번 곱한 뒤, C로 나눈 나머지는?

제한 사항

- A, B, C는 1 <= A, B, C <= 2,147,483,647 (int의 최댓값)

예제 입력

```
10 11 12
```

예제 출력

```
4
```

1629번 : 곱셈 – Silver 1

문제

- 자연수 A를 B번 곱한 뒤, C로 나눈 나머지는?

제한 사항

- A, B, C는 1 <= A, B, C <= 2,147,483,647 (int의 최댓값)

Divide : 제곱 수 나누기
Conquer : B가 1인가?
Combine : 곱한 결과들 합친 뒤, C로 나눈 나머지 구하기

예제 입력

```
10 11 12
```

예제 출력

```
4
```

2630번 : 색종이 만들기 – Silver 3

문제

- 파란색 또는 하얀색이 칠해진 색종이가 주어진다
- 규칙에 따라 색종이를 자른다
  : 전체 종이가 모두 같은 색으로 칠해진 것이 아니라면 같은 크기로 4등분 한다
- 다양한 크기를 가진 정사각형 모양의 하얀색 또는 파란색 색종이의 개수는?

제한 사항

- $N(=2^k)$ $1 <= k <= 7$

/<> 2630번 : 색종이 만들기 – Silver 3

문제

- 파란색 또는 하얀색이 칠해진 색종이가 주어진다
- 규칙에 따라 색종이를 자른다
  : 전체 종이가 모두 같은 색으로 칠해진 것이 아니라면 같은 크기로 4등분 한다
- 다양한 크기를 가진 정사각형 모양의 하얀색 또는 파란색 색종이의 개수는?

제한 사항

- N(=2^k) $1 <= k <= 7$

Divide : 색종이 나누기
Conquer : 부분 종이가 모두 같은 색인가?
Combine : 색종이의 개수 세기

예제

예제 입력

```
8
1 1 0 0 0 0 1 1
1 1 0 0 0 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 0
1 0 0 0 1 1 1 1
0 1 0 0 1 1 1 1
0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
```

예제 출력

```
9
7
```

일단 잘라볼까요?

#알튜비튜

1	1	0	0	0	0	1	1
1	1	0	0	0	0	1	1
0	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
1	0	0	0	1	1	1	1
0	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

일단 잘라볼까요?

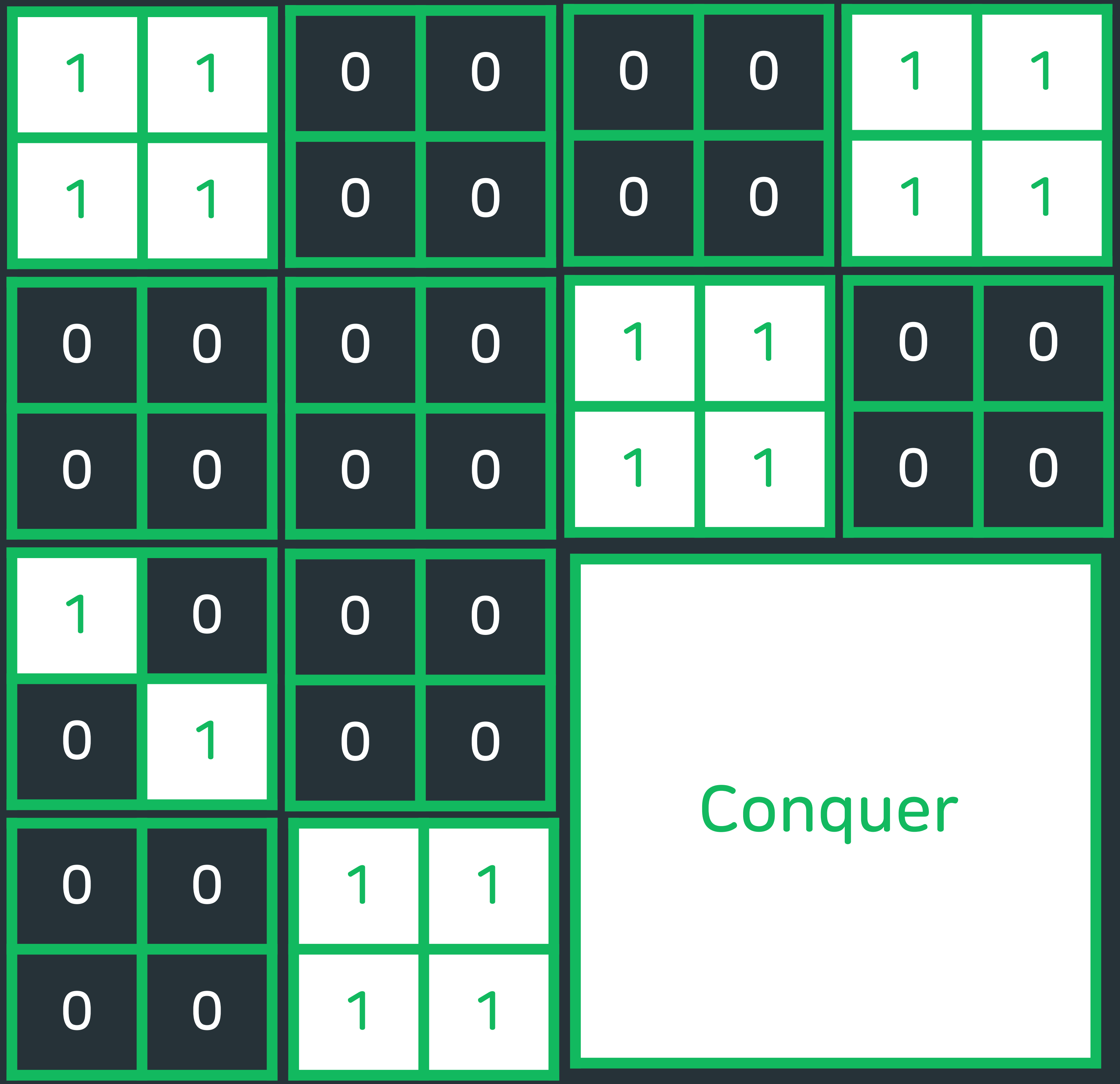
1	1	0	0
1	1	0	0
0	0	0	0
0	0	0	0

0	0	1	1
0	0	1	1
1	1	0	0
1	1	0	0

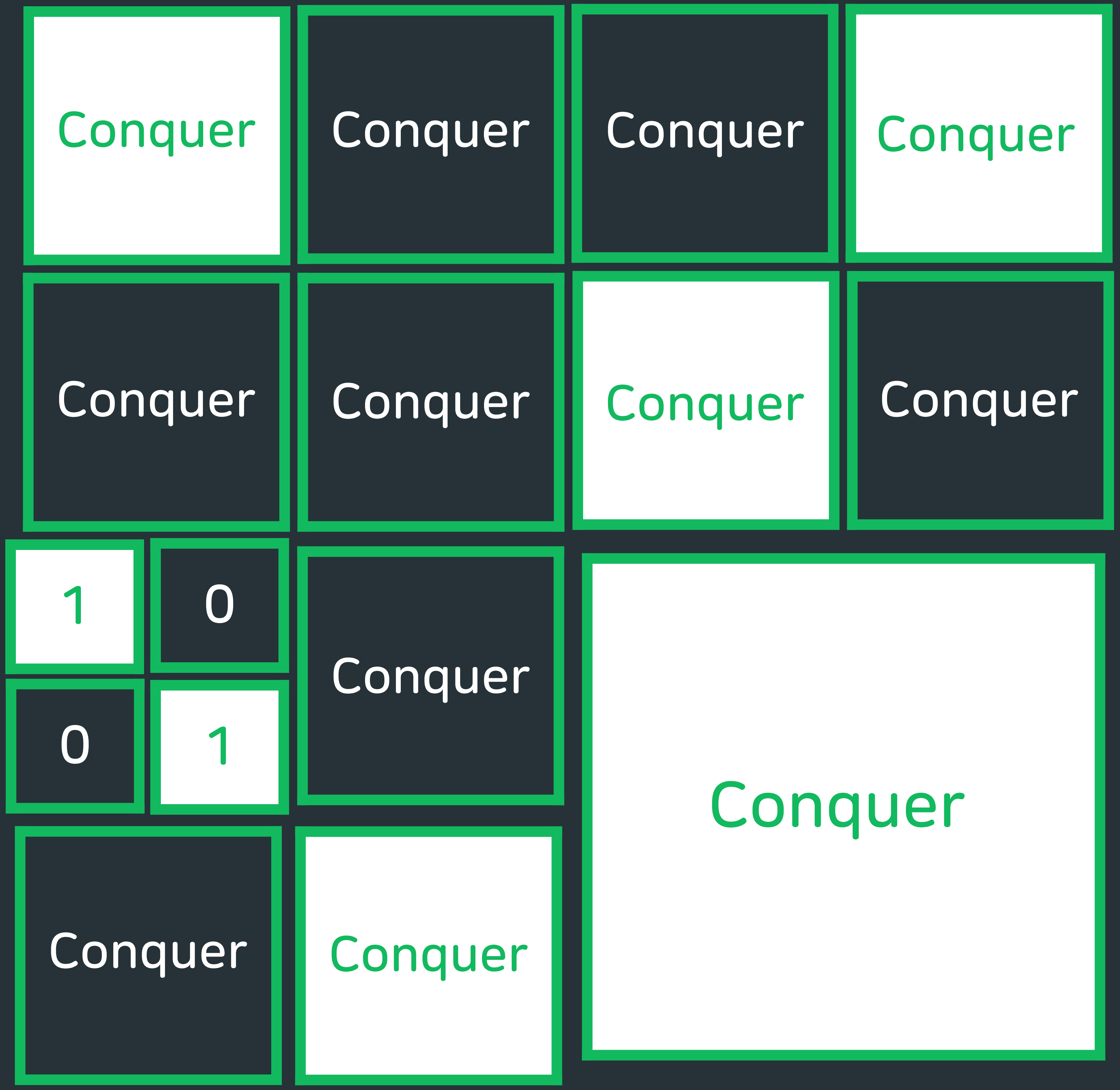
1	0	0	0
0	1	0	0
0	0	1	1
0	0	1	1

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

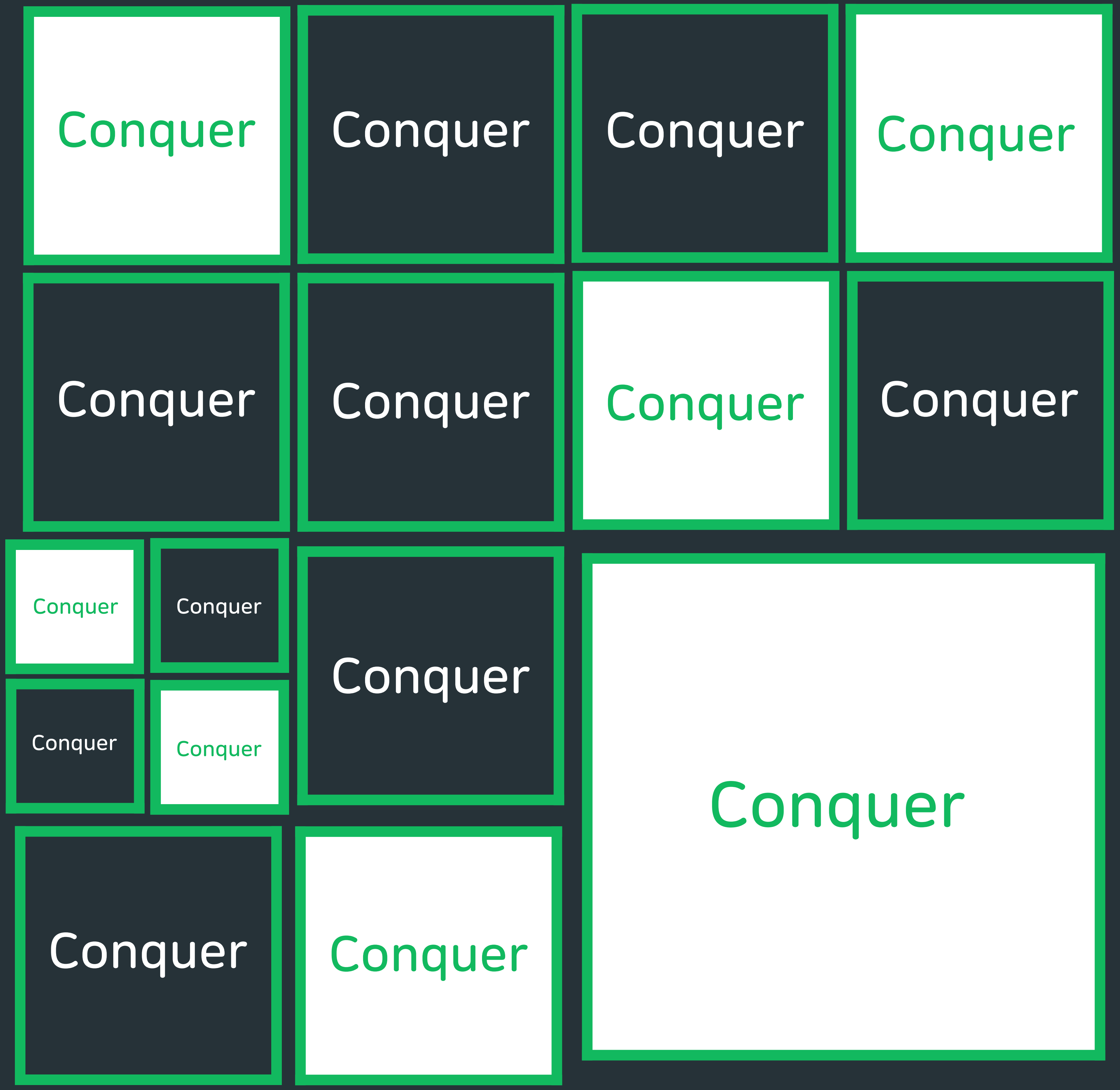
일단 잘라볼까요?



일단 잘라볼까요?


Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
1
0
0
1
Conquer
Conquer
Conquer
Conquer
Conquer

일단 잘라볼까요?


Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer
Conquer

10830번 : 행렬 제곱 – Gold 4

문제

- 크기가 N*N인 행렬 A를 B제곱한 결과는?

제한 사항

- N의 범위는 2 <= N <= 5
- B의 범위는 1 <= B <= 100,000,000,000 (1000억)
- 행렬 A의 원소 k는 0 <= k <= 1000

10830번 : 행렬 제곱 – Gold 4

문제

- 크기가 N*N인 행렬 A를 B제곱한 결과는?

제한 사항

- N의 범위는 2 <= N <= 5
- B의 범위는 1 <= B <= 100,000,000,000 (1000억)
- 행렬 A의 원소 k는 0 <= k <= 1000

Divide : 제곱 수 나누기
Conquer : B가 1인가?
Combine : 곱한 결과들 합치기

예제

#알튜비튜

예제 입력 1

```
2 5
1 2
3 4
```

예제 출력 1

```
69 558
337 406
```

예제 입력 2

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

예제 출력 2

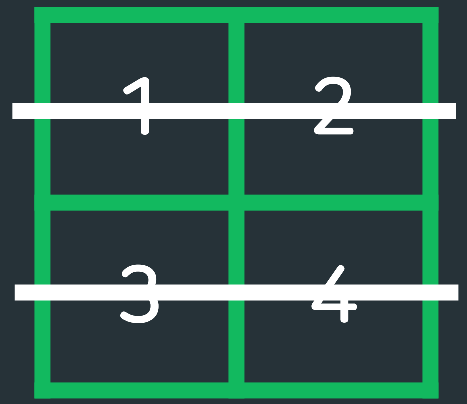
```
468 576 684
62 305 548
656 34 412
```

몰래 보세요

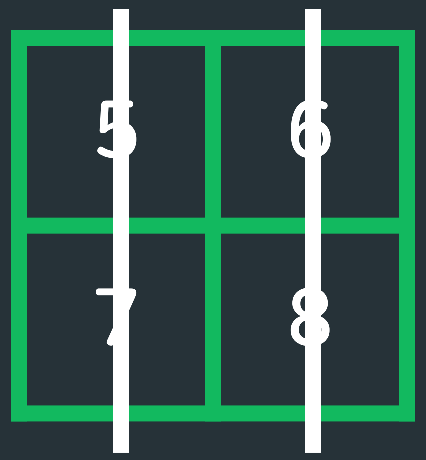
Hint

1. 혹시 98년 혹은 그보다 늦게 태어나셨다면… 빨리 구글에 행렬 곱셈을 검색해봅시다
2. 어떤 수를 제곱하는 문제는 아까도 봤었어요!

행렬 곱셈

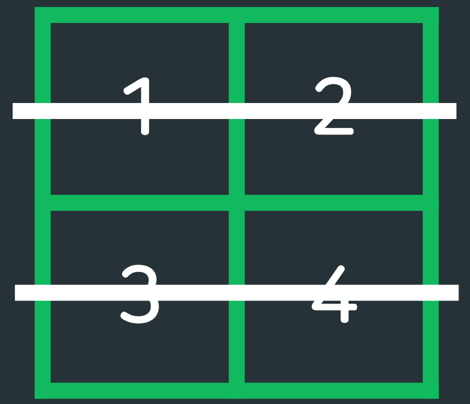


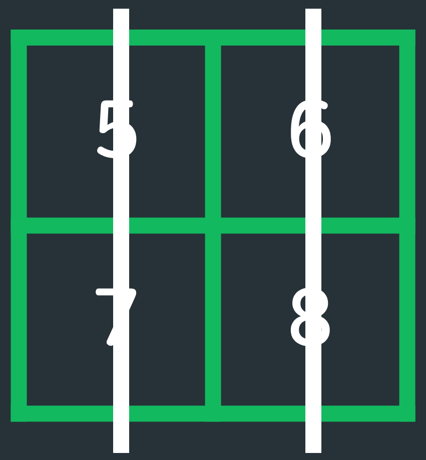
행 (row)



열 (col)

행렬 곱셈





행 (row) X 열 (col)

행렬 곱셈

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \times \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} = \begin{bmatrix}? & \\ & \end{bmatrix}$$

행렬 곱셈

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 5 & \\ & \end{bmatrix}$$

행렬 곱셈

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & \\ & \end{bmatrix}$$

행렬 곱셈

$$
\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ & \end{bmatrix}
$$

행렬 곱셈

$$\begin{bmatrix} 1 & 2 \\ \cancel{3} & \cancel{4} \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & \end{bmatrix}$$

행렬 곱셈

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \times \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

4256번 : 트리 - Gold 4

문제

- 노드의 개수가 n인 이진 트리를 전위 순회, 중위 순회 한 결과가 주어진다.
- 이진 트리를 후위 순회한 결과는?

제한 사항

- n의 범위는 1 <= n <= 1,000

예제

예제 입력

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

예제 출력

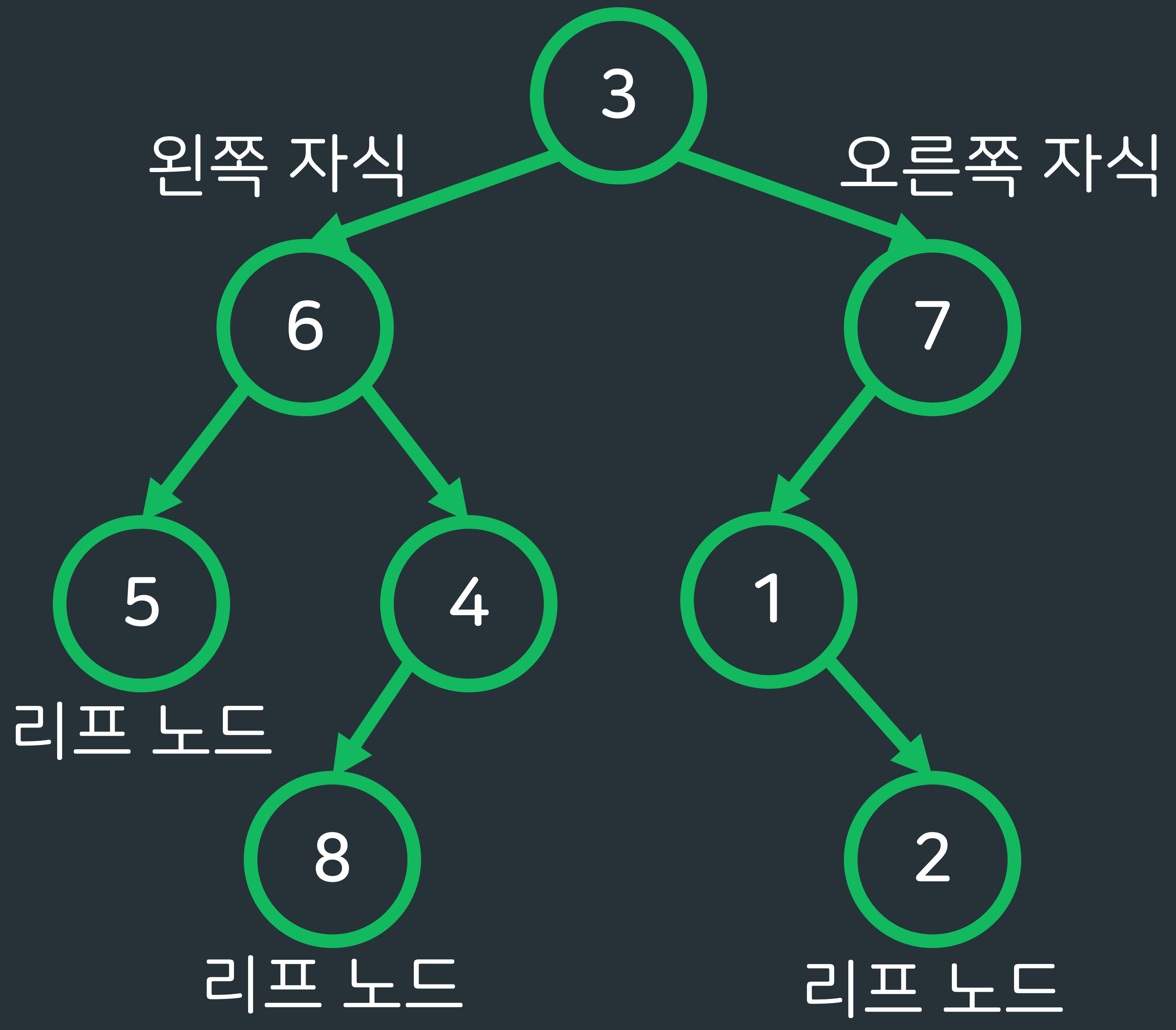
```
2 4 1 3
5 8 4 6 2 1 7 3
```

몰래 보세요

Hint

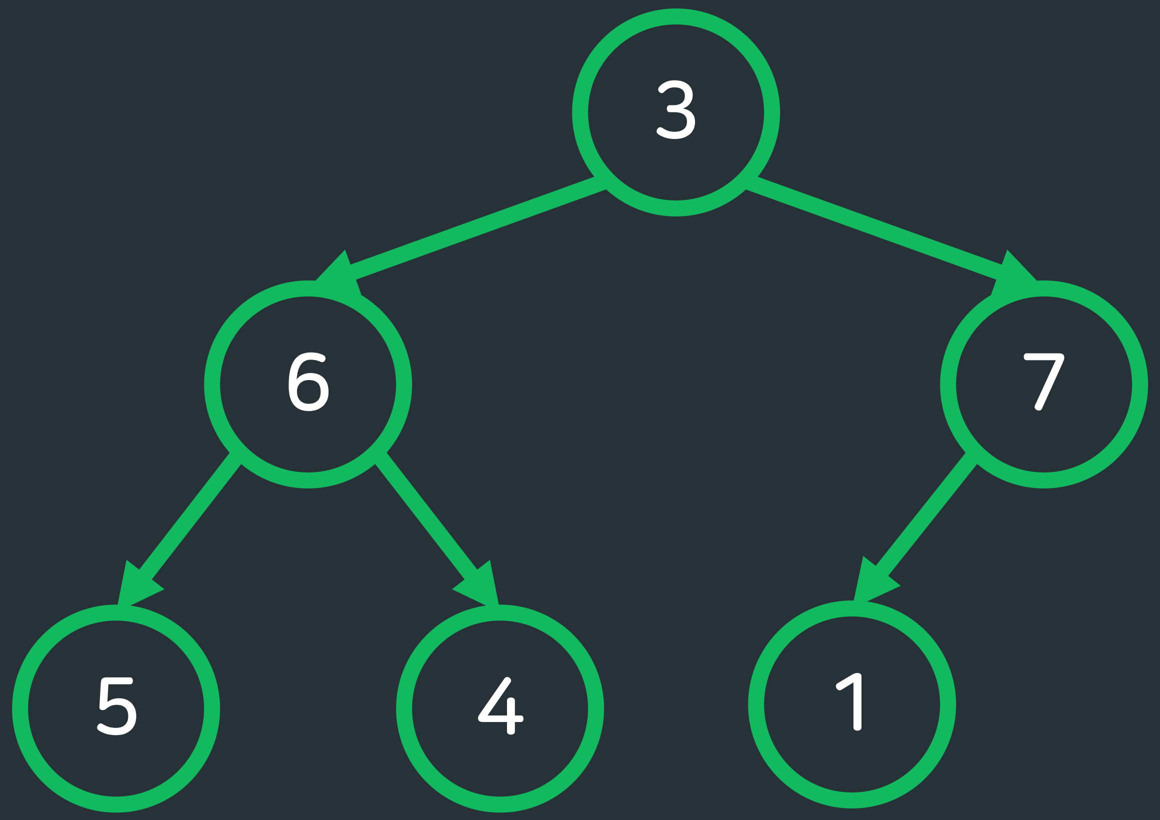
1. 이건 '분할 정복' 문제 입니다! 트리에 집중하지 말고 분할할 수 있는 부분에 집중해봐요
2. 의사 코드에 의하면 트리를 세 부분으로 분할할 수 있어요

이진 트리



모든 노드가 최대 2개의 자식 노드가 있는 트리

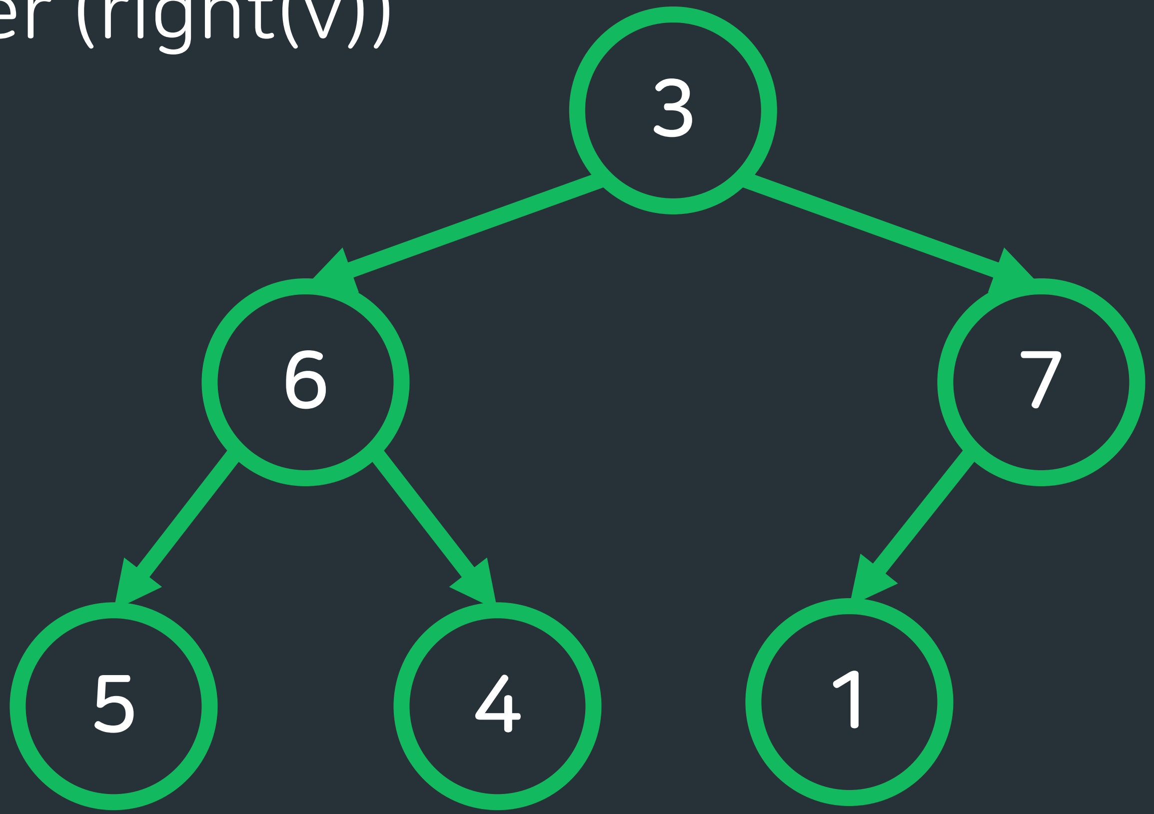
전위 순회




```
preorder (v)
{
  if (v != null)
    print (v)
    preorder (left(v))
    preorder (right(v))
}
```

전위 순회

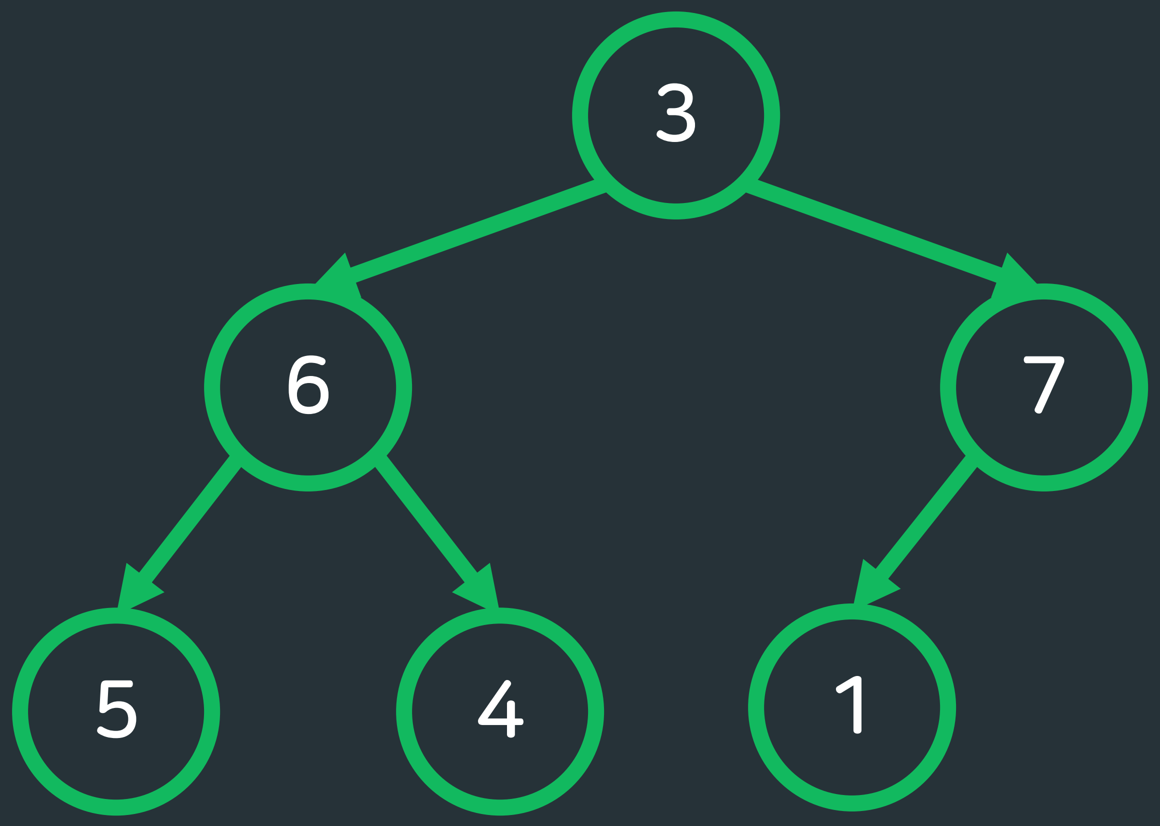

```
preorder (v)
{
  if (v != null)
    print (v)
    preorder (left(v))
    preorder (right(v))
}
```



3 6 5 4 7 1

```
preorder(3)
  print(3)
  preorder(3 -> left : 6)
    print(6)
    preorder(6 -> left : 5)
      print(5)
      preorder(5 -> left : null)
      preorder(5 -> right : null)
    preorder(6-> right : 4)
      print(4)
      preorder(4 -> left : null)
      preorder(4 -> right : null)
  preorder(3 -> right : 7)
    print(7)
    preorder(7 -> left : 1)
      print(1)
      preorder(1 -> left : null)
      preorder(1 -> right : null)
    preorder(7 -> right : null)
```

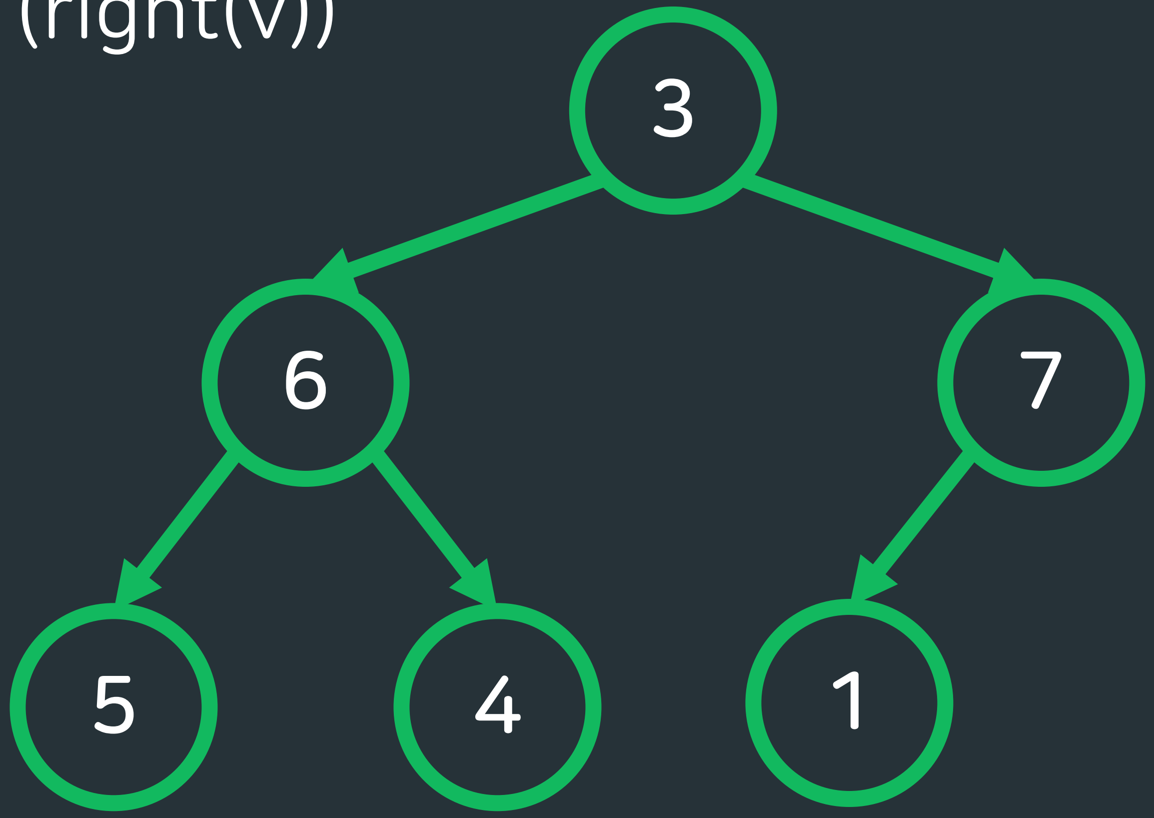
중위 순회




```
inorder (v)
{
  if (v != null)
    inorder (left(v))
    print (v)
    inorder (right(v))
}
```

중위 순회

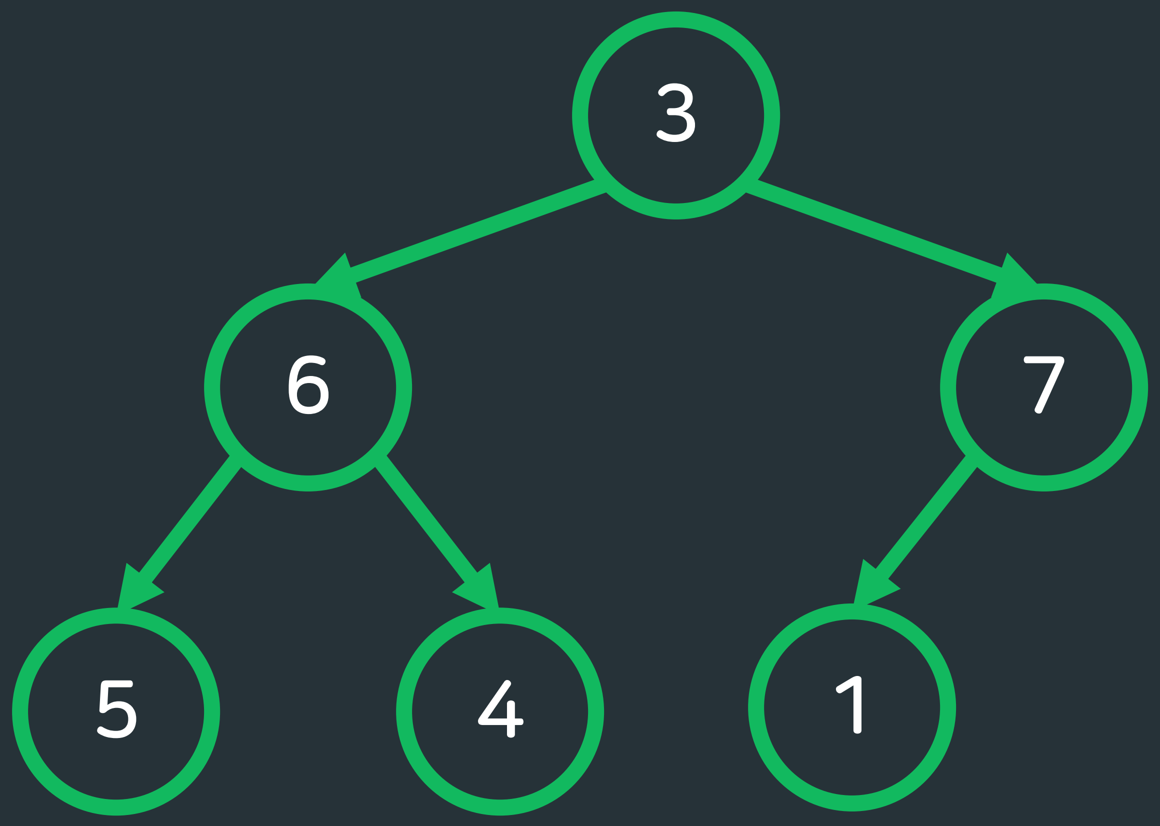

```
inorder (v)
{
  if (v != null)
    inorder (left(v))
    print (v)
    inorder (right(v))
}
```



5 6 4 3 1 7

```
inorder(3)
  inorder(3 -> left : 6)
    inorder(6 -> left : 5)
      inorder(5 -> left : null)
      print(5)
      inorder(5 -> right : null)
    print(6)
    inorder(6-> right : 4)
      inorder(4 -> left : null)
      print(4)
      inorder(4 -> right : null)
  print(3)
  inorder(3 -> right : 7)
    inorder(7 -> left : 1)
      inorder(1 -> left : null)
      print(1)
      inorder(1 -> right : null)
    print(7)
    inorder(7 -> right : null)
```

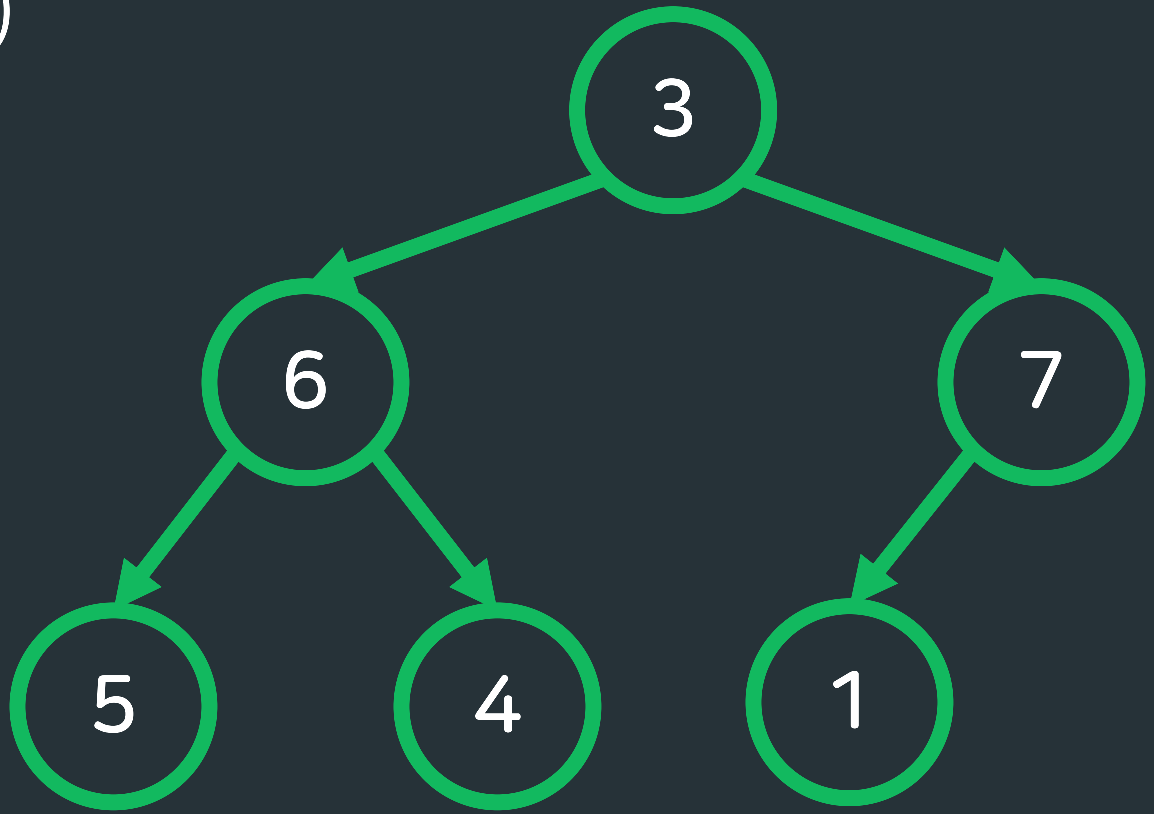
후위 순회




```
postorder (v)
{
  if (v != null)
    postorder (left(v))
    postorder (right(v))
    print (v)
}
```

후위 순회

```
postorder (v)
{
  if (v != null)
    postorder (left(v))
    postorder (right(v))
    print (v)
}
```



5 4 6 1 7 3

```
postorder(3)
  postorder(3 -> left : 6)
    postorder(6 -> left : 5)
      postorder(5 -> left : null)
      postorder(5 -> right : null)
      print(5)
    postorder(6-> right : 4)
      postorder(4 -> left : null)
      postorder(4 -> right : null)
      print(4)
    print(6)
  postorder(3 -> right : 7)
    postorder(7 -> left : 1)
      postorder(1 -> left : null)
      postorder(1 -> right : null)
      print(1)
    postorder(7 -> right : null)
    print(7)
  print(3)
```

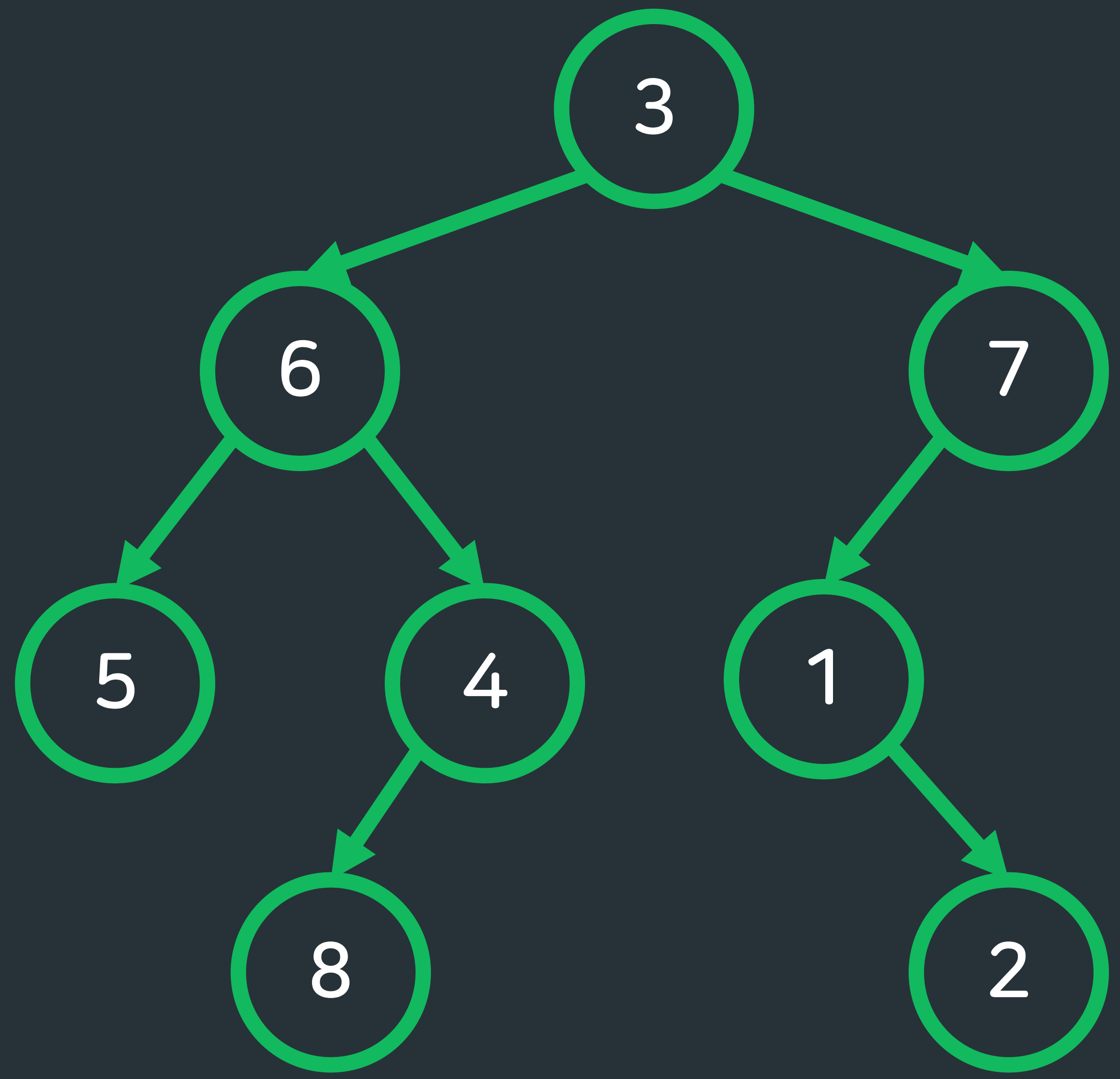
어떻게 분할하지?

preorder

3	6	5	4	8	7	1	2

inorder

5	6	8	4	3	1	2	7



어떻게 분할하지?

preorder

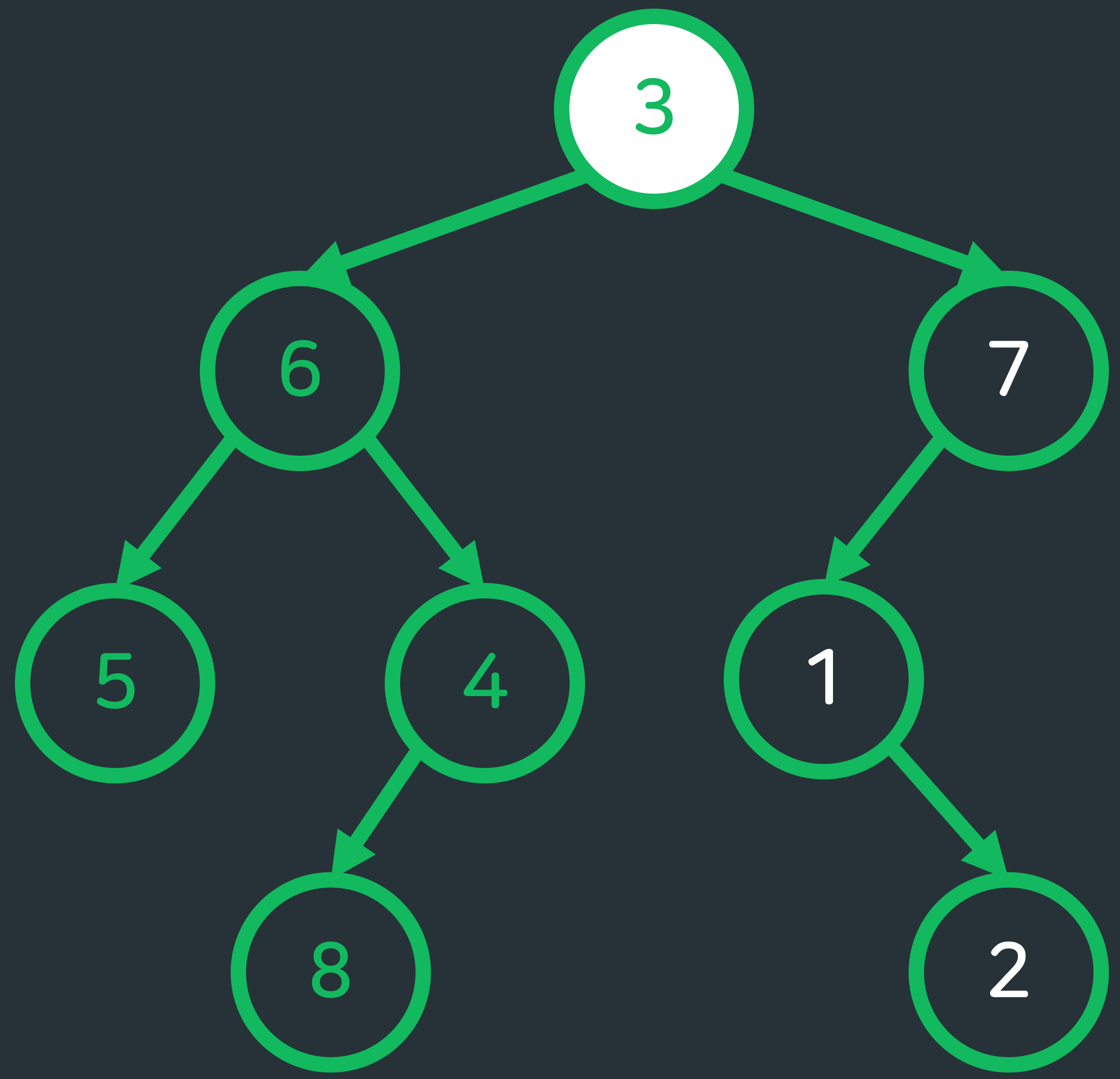
3	6	5	4	8	7	1	2

root left right

inorder

5	6	8	4	3	1	2	7

left root right



어떻게 분할하지?

preorder

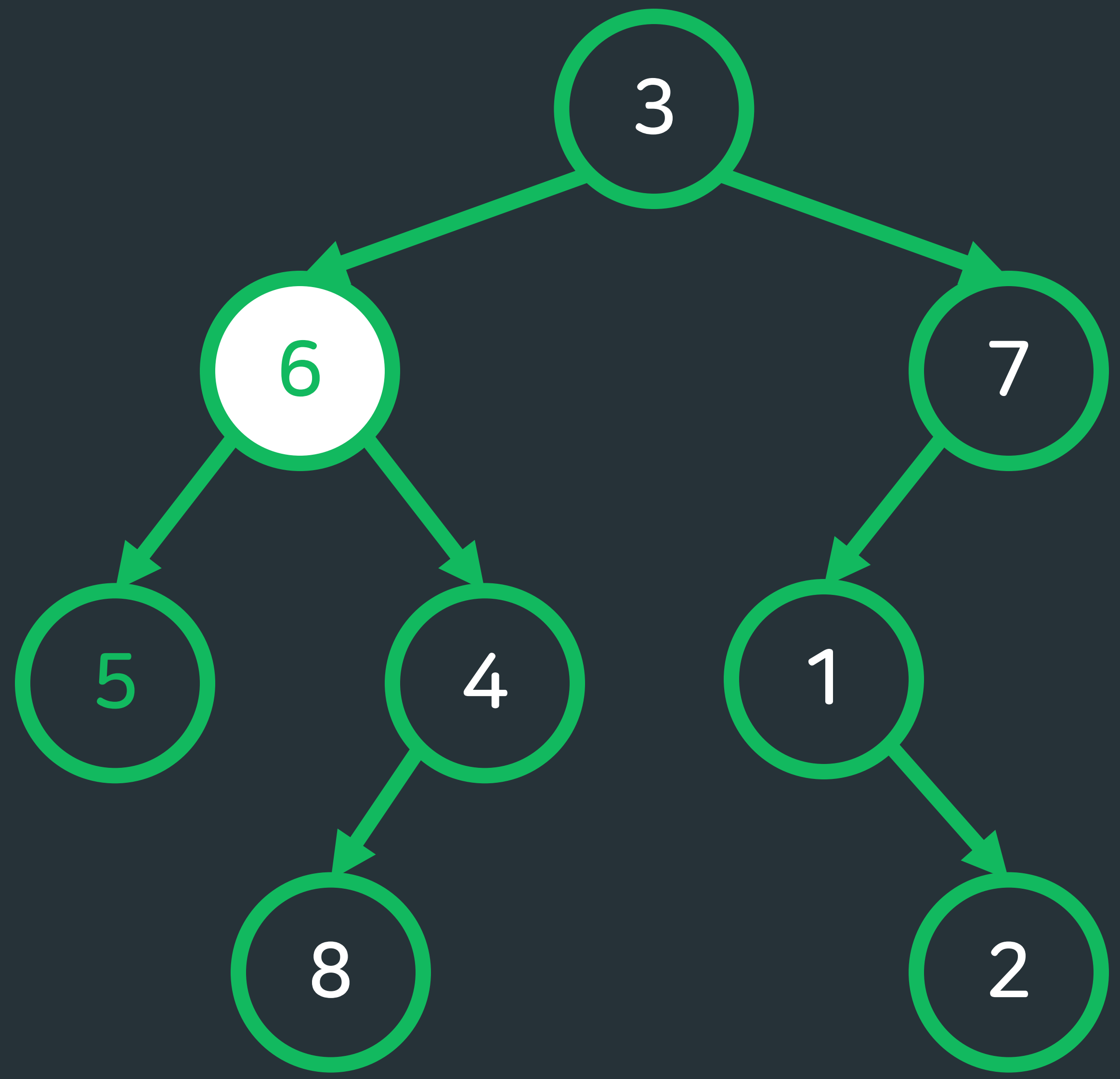
3	6	5	4	8	7	1	2

root left right

inorder

5	6	8	4	3	1	2	7

left root right



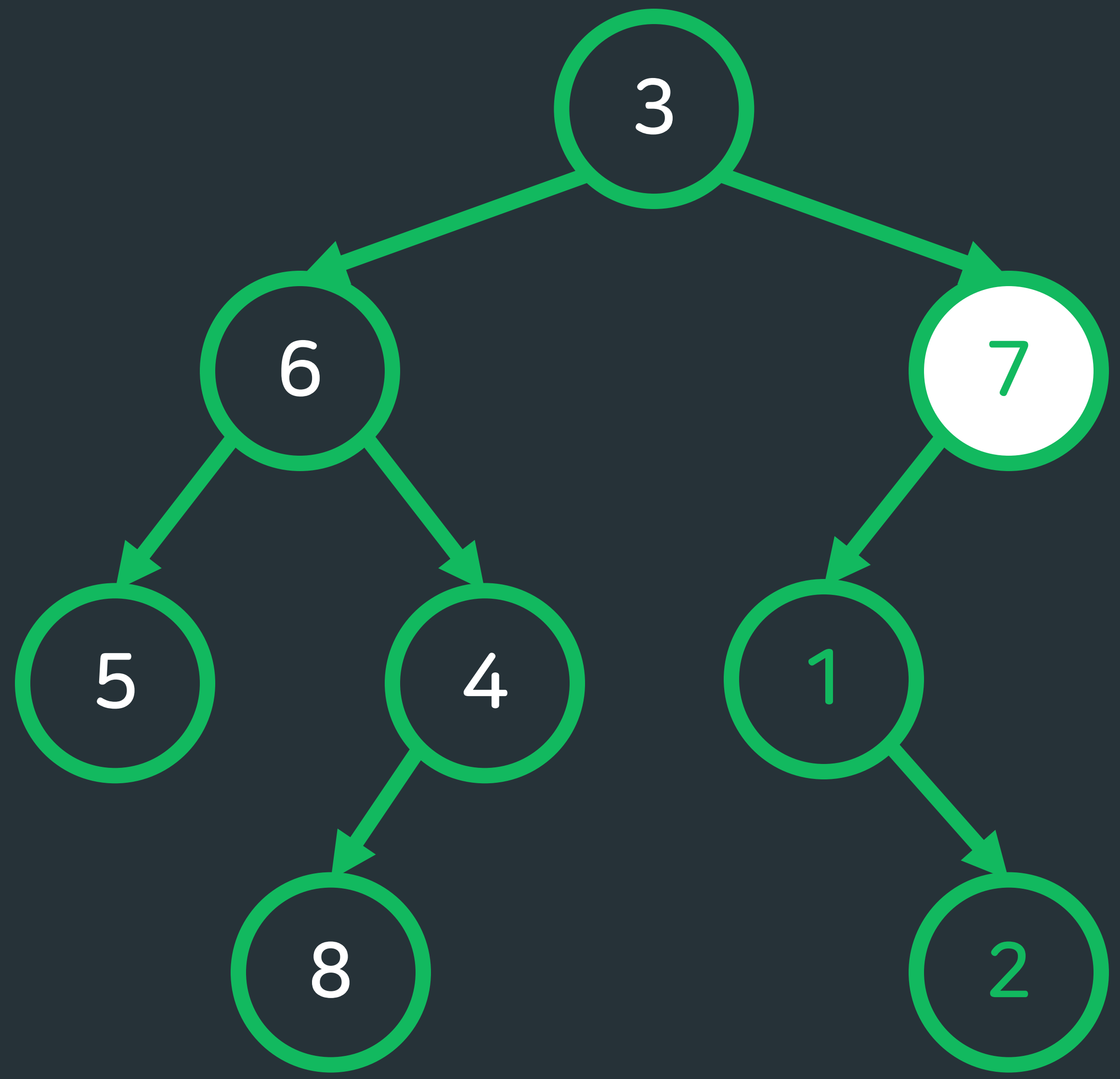
어떻게 분할하지?

preorder

3	6	5	4	8	7	1	2
					root		left

inorder

5	6	8	4	3	1	2	7
						left	root



결론

#알튜비튜

Preorder

- 부분 트리의 root는 해당 부분 트리 구간의 맨 왼쪽에 있다.
- root 이후에 왼쪽 서브 트리의 노드와 오른쪽 서브 트리의 노드가 이어진다.
  -> root의 위치는 알 수 있지만, 왼쪽/오른쪽 서브 트리가 나뉘는 경계는 알 수 없음

Inorder

- 부분 트리의 root를 기준으로 왼쪽에는 왼쪽 서브 트리가, 오른쪽에는 오른쪽 서브 트리가 위치한다.
  -> root의 위치만 알면 왼쪽, 오른쪽 서브 트리의 노드를 알 수 있지만 root의 위치를 알 수 없음

결론

Preorder

- 부분 트리의 root는 해당 부분 트리 구간의 맨 왼쪽에 있다.
- root 이후에 왼쪽 서브 트리의 노드와 오른쪽 서브 트리의 노드가 이어진다.
  -> root의 위치는 알 수 있지만, 왼쪽/오른쪽 서브 트리가 나뉘는 경계는 알 수 없음

Inorder

- 부분 트리의 root를 기준으로 왼쪽에는 왼쪽 서브 트리가, 오른쪽에는 오른쪽 서브 트리가 위치한다.
  -> root의 위치만 알면 왼쪽, 오른쪽 서브 트리의 노드를 알 수 있지만 root의 위치를 알 수 없음

Preorder로 루트 노드를 파악하고,
Inorder로 왼쪽 오른쪽 서브 트리를 나누자!

의사 코드

```
void divide(전위_순회에서_루트_위치, 서브_트리의_왼쪽_경계(중위), 서브_트리의_오른쪽_경계(중위)){
    if(왼쪽_경계>오른쪽_경계)
        return;

    루트의_값 = '전위_순회에서_루트_위치'의 값
    중위_순회에서_루트_위치 = 중위 순회에서 '루트의_값'이 위치한 인덱스

    //왼쪽 서브트리
    divide(전위_순회에서_루트_위치 + 1, 서브_트리의_왼쪽_경계, 중위_순회에서_루트_위치 - 1)

    //오른쪽 서브트리
    /**
     * 왼쪽_서브_트리의_루트_위치 = 전위_순회에서_루트_위치 + 1
     * 왼쪽_서브_트리의_크기 = 중위_순회에서_루트_위치 - 서브_트리의_왼쪽_경계
     */
    divide(왼쪽_서브_트리의_루트_위치 + 왼쪽_서브_트리의_크기, 중위_순회에서_루트_위치 + 1, 서브_트리의_오른쪽_경계)

    print(루트의_값)
}
```

preorder

3	6	5	4	8	7	1	2

inorder

5	6	8	4	3	1	2	7

마무리

정리

- 문제에 반복되는 연산이 보이거나 (ex 별 찍기) N이 아주 크다면 분할 정복 생각하기!
- 단순 계산 문제라면 N은 long long 범위까지 들어올 수 있음
- Divide, Conquer, Combine 연산이 어떤 것일지 먼저 생각하기
- 구현 방법에 따라 시간 복잡도가 다양. 잘못 구현했다면, 분할 정복임에도 시간초과가 발생할 수 있음
- 재귀 함수를 구현할 때에는 무한루프에 빠지지 않도록 기저조건을 확실히 하기

이것도 알아보세요!

- 입력이 이미 또는 거의 정렬된 상태라면, Quick sort 효율성이 떨어지는 건 확인했습니다!
  그렇다면, 이런 상황에서 Quick sort의 효율성을 확보하기 위해서는 어떻게 해야 할까요?

과제

필수

- 17281번 : ⚾ - Gold 4
- 1244번 : 스위치 켜고 끄기 – Silver 4

3문제 이상 선택

- 1074번 : Z – Silver 1
- 1802번 : 종이 접기 – Silver 2
- 2447번 : 별 찍기 – 10 – Silver 1
- 17829번 : 222-풀링 – Silver 2
- 16198번 : 에너지 모으기 – Silver 1
- 21314번 : 민겸 수 – Silver 2