Chapter 07.
분할 정복

Clip 01 | [1629] 곱셈
빠른 거듭제곱

Clip 02 | [1074] Z
좌표평면의 분할과 탐색

Clip 03 | [1992] 쿼드트리
영역의 분할과 압축

Clip 04 | [1780] 종이의 개수
영역 분할의 확장

Clip 05 | [2751] 수 정렬하기 2
합병 정렬의 구현

분할 정복이란?

하나의 커다란 문제를 작은 부분문제로 나누어 해결하고

결과를 결합하여, 원래 문제의 해를 구하는 알고리즘

존 폰 노이만

- 1945년에 헤르만 골드스타인과 함께, 분할 정복을 이용한 합병 정렬을 고안하였다
- 컴퓨터 역사에서 큰 이벤트가 발생했을 때 이 사람을 지목하면 대충 맞는다.

분할 정복이란?

앞서 다루었던 재귀

하나의 커다란 문제를 작은 부분문제로 나누어 해결하고

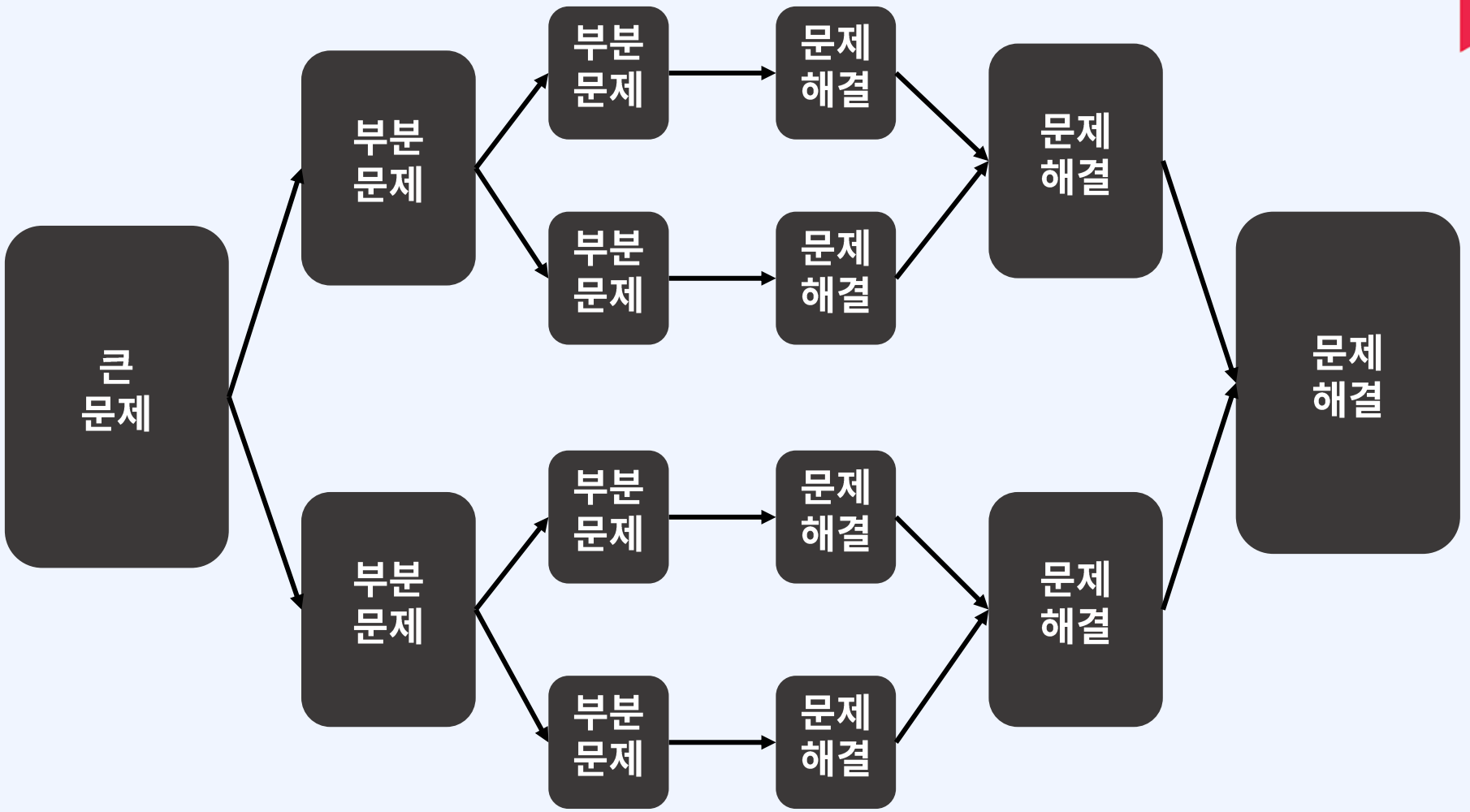
결과를 결합하여, 원래 문제의 해를 구하는 알고리즘

새로 추가된 부분

아래 3가지 단계로 문제를 처리한다

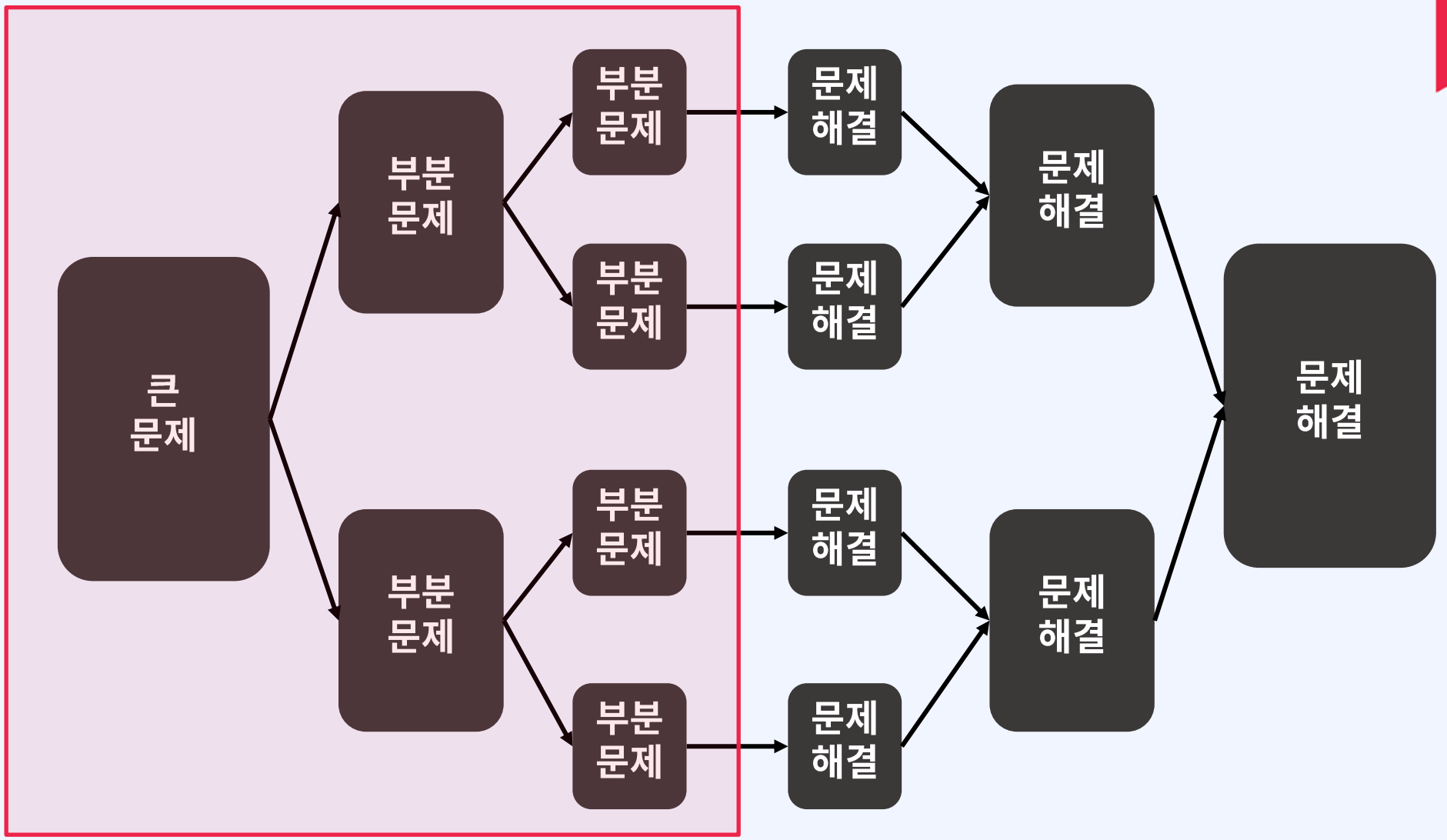
- 분할 (Divide)
- 정복 (Conquer)
- 조합 (Combine)

분할 정복이란?



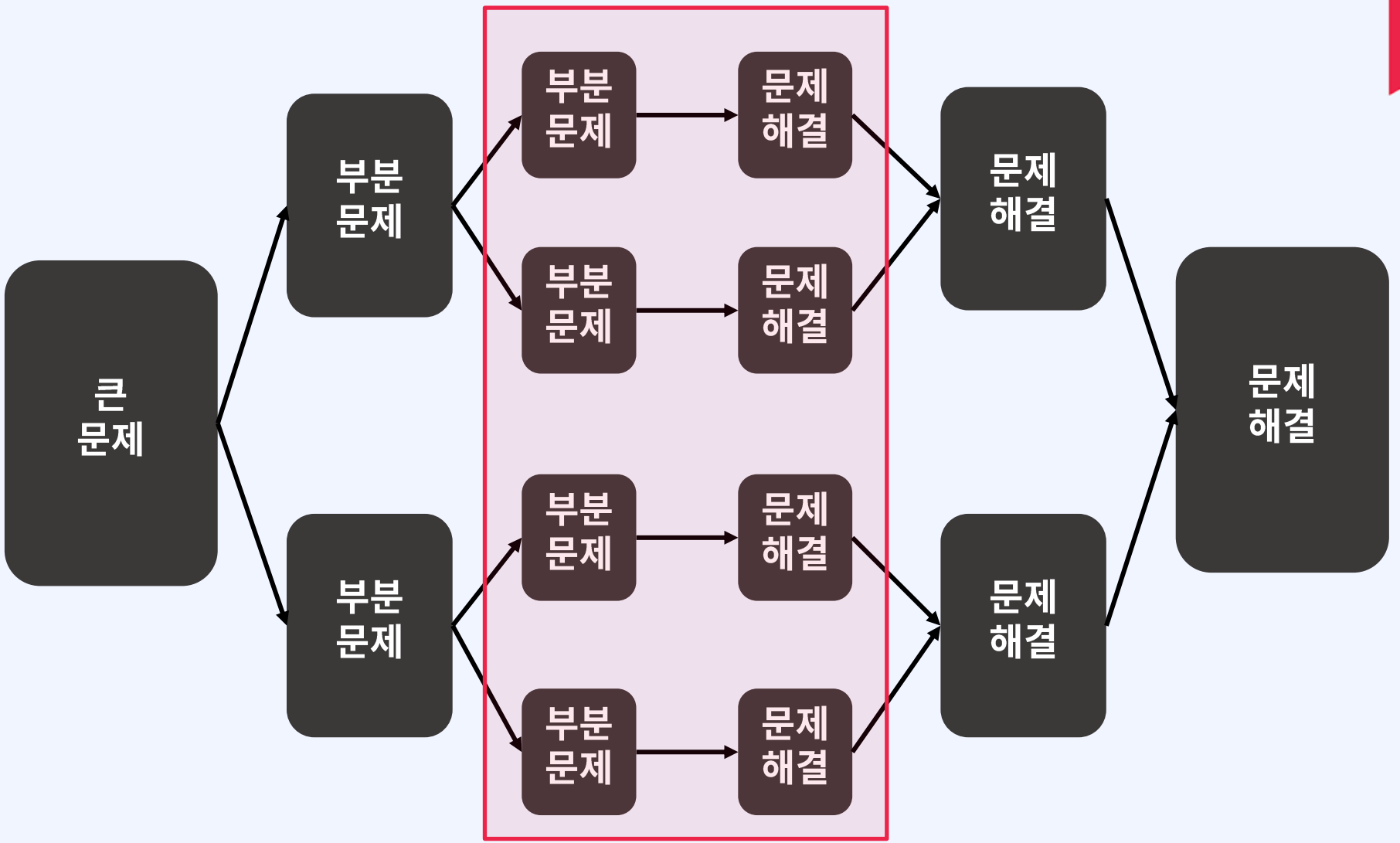
분할 정복이란?

1. 분할 (Divide)



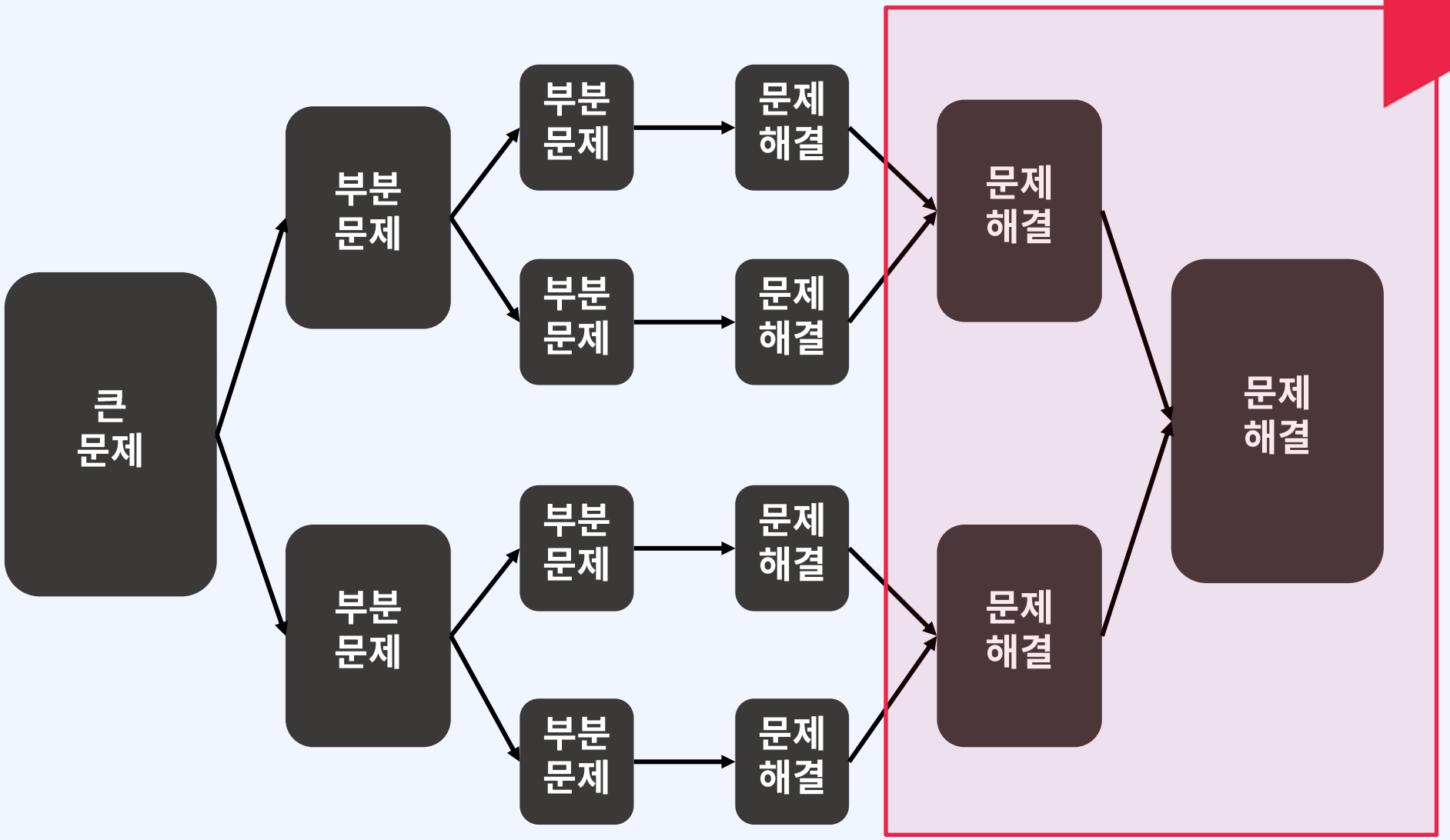
분할 정복이란?

2. 정복 (Conquer)



분할 정복이란?

3. 조합 (Combine)



분할 정복이란?

분할 (Divide)

- 문제를 손쉽게 해결 가능한 수준까지 부분 문제로 나눈다

정복 (Conquer)

- 부분 문제를 해결(정복) 한다.

조합 (Combine)

- 하위 문제의 결과를 합쳐 큰 문제로 반환 한다

BOJ1629: 곱셈

곱셈

☆

문제

$A^B \ (MOD\ C)$

자연수 A를 B번 곱한 수를 알고 싶다. 단 구하려는 수가 매우 커질 수 있으므로 이를 C로 나눈 나머지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 A, B, C가 빈 칸을 사이에 두고 순서대로 주어진다. A, B, C는 모두 2,147,483,647 이하의 자연수이다.

출력

첫째 줄에 A를 B번 곱한 수를 C로 나눈 나머지를 출력한다.

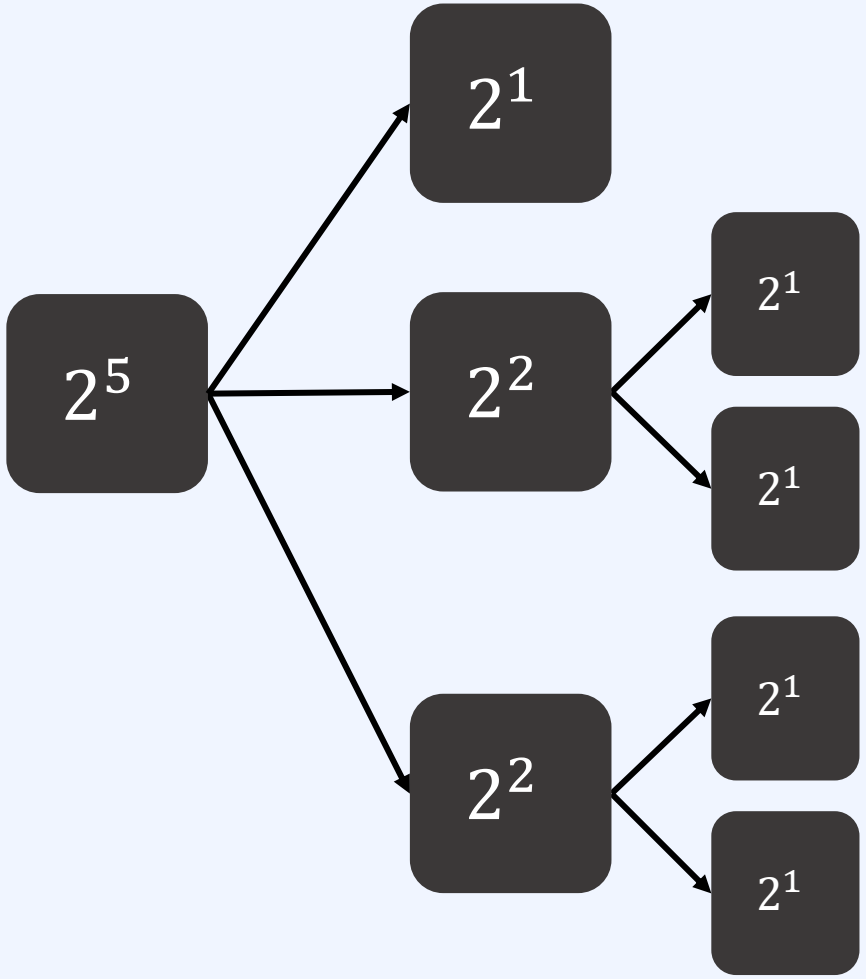
예제 입력 1 복사

```
10 11 12
```

예제 출력 1 복사

```
4
```

BOJ1629: 곱셈



알아 두어야할 사전 지식

- 지수함수의 곱
  - $2^n * 2^m = 2^{n+m}$
- 나머지 연산의 분배법칙
  - (A * B) % C = (A % C) * (B % C)

BOJ1629: 곱셈

a^b

```
public static long power(long a, long b) { }
```

b가 짝수인 경우

- power(a, b) → power(a, b/2) * power(a, b/2)

b가 홀수인 경우

- power(a, b) → power(a, b/2) * power(a, b/2) * power(a, 1)

연산은 한번하고 재활용

BOJ1629: 곱셈

b가 짝수인 경우

- power(a, b) → power(a, b/2) * power(a, b/2)

연산은 한번하고 재활용

b가 홀수인 경우

- power(a, b) → power(a, b/2) * power(a, b/2) * power(a, 1)

```
public static long power(long a, long b) {
    if(b == 1) return a % c;
    long half = power(a, b / 2);
    if(b % 2 == 0) {
        return (half * half) % c;
    } else {
        return (((half * half) % c) * a) % c;
    }
}
```

분할 정복이란?

분할 (Divide)

- 문제를 손쉽게 해결 가능한 수준까지 부분 문제로 나눈다

정복 (Conquer)

- 부분 문제를 해결(정복) 한다.

조합 (Combine)

- 하위 문제의 결과를 합쳐 큰 문제로 반환 한다

BOJ1629: 곱셈

분할 (Divide)

- 문제를 손쉽게 해결 가능한 수준까지 부분 문제로 나눈다

정복 (Conquer)

- 부분 문제를 해결(정복) 한다.

조합 (Combine)

- 하위 문제의 결과를 합쳐 큰 문제로 반환 한다

```
public static long power(long a, long b) {
    if(b == 1) return a % c;
    long half = power(a, b / 2);
    if(b % 2 == 0) {
        return (half * half) % c;
    } else {
        return (((half * half) % c) * a) % c;
    }
}
```

BOJ1629: 곱셈

분할 (Divide)
- 문제를 손쉽게 해결 가능한 수준까지 부분 문제로 나눈다

정복 (Conquer)
- 부분 문제를 해결(정복) 한다.

조합 (Combine)
- 하위 문제의 결과를 합쳐 큰 문제로 반환 한다

```
public static long power(long a, long b) {
  if(b == 1) return a % c;
  long half = power(a, b / 2);
  if(b % 2 == 0) {
    return (half * half) % c;
  } else {
    return (((half * half) % c) * a) % c;
  }
}
```

BOJ1629: 곱셈

분할 (Divide)
- 문제를 손쉽게 해결 가능한 수준까지 부분 문제로 나눈다

정복 (Conquer)
- 부분 문제를 해결(정복) 한다.

조합 (Combine)
- 하위 문제의 결과를 합쳐 큰 문제로 반환 한다

```
public static long power(long a, long b) {
    if(b == 1) return a % c;
    long half = power(a, b / 2);
    if(b % 2 == 0) {
        return (half * half) % c;
    } else {
        return (((half * half) % c) * a) % c;
    }
}
```

Ch07. 분할 정복

2. [1074] Z

BOJ1074: Z

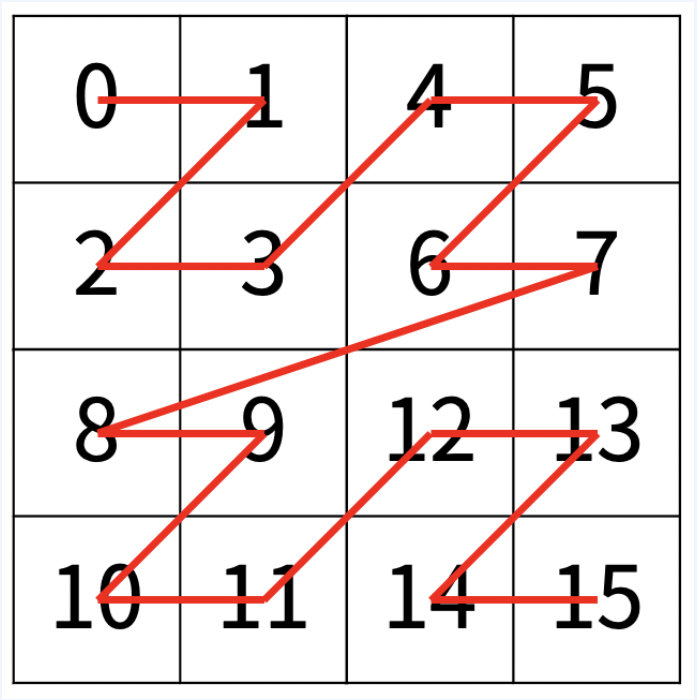
문제 요약

- 2^n * 2^n 2차원 배열을 Z모양으로 탐색
  - (좌|상단) → (우|상단) → (좌|하단) → (우|하단)

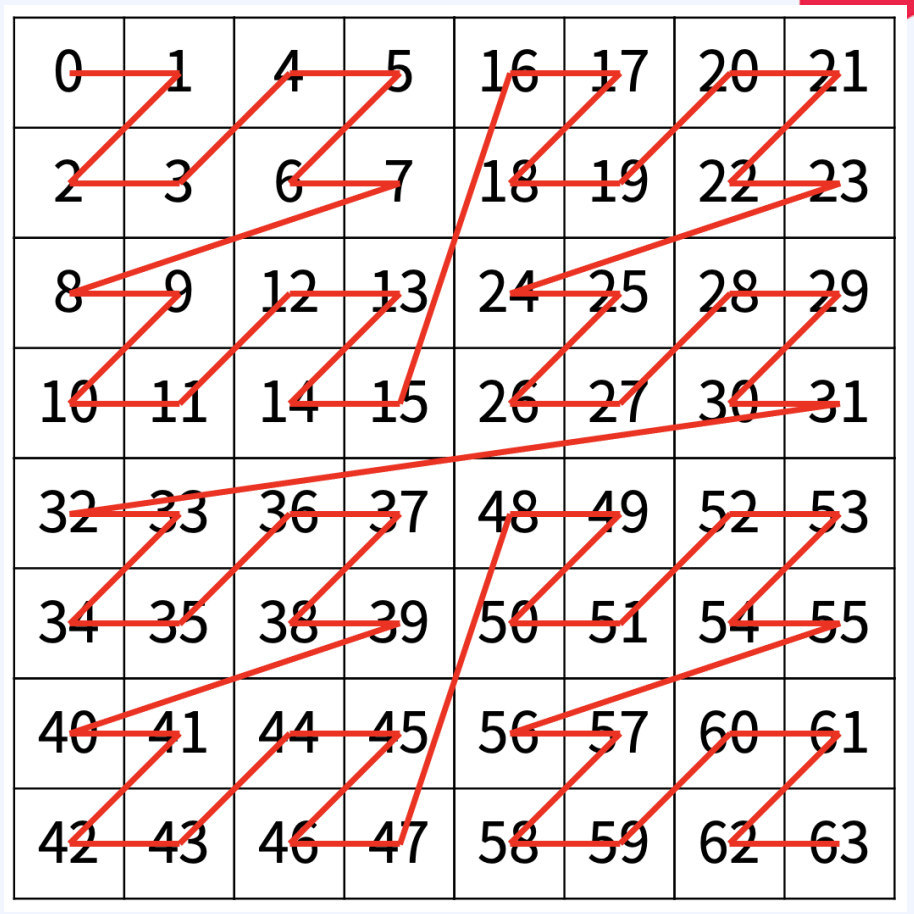
- N이 주어 졌을 때 (r, c)의 좌표는
  몇 번째로 방문하는지 출력

- 0 <= N <= 15
- 0 <= r, c < 2^N

BOJ1074: Z



N = 2

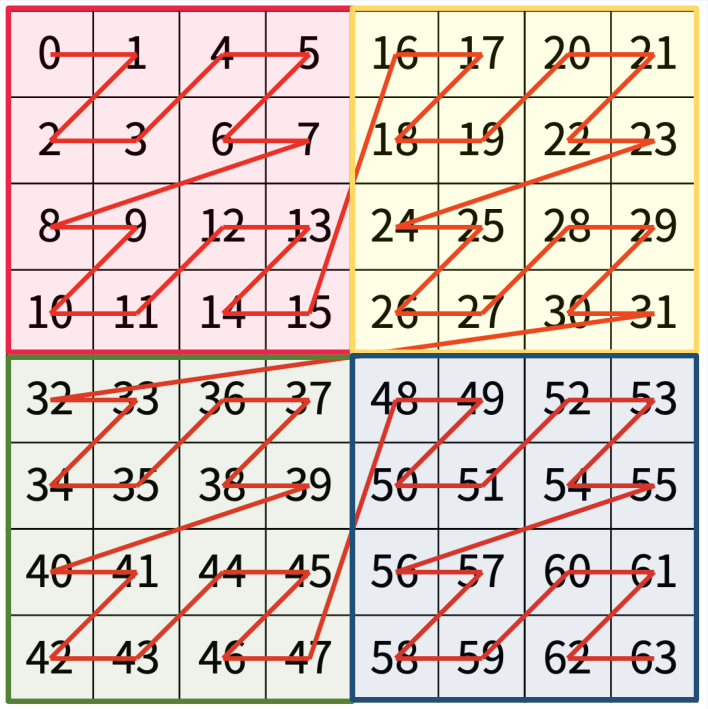


N = 3

BOJ1074: Z

문제 접근

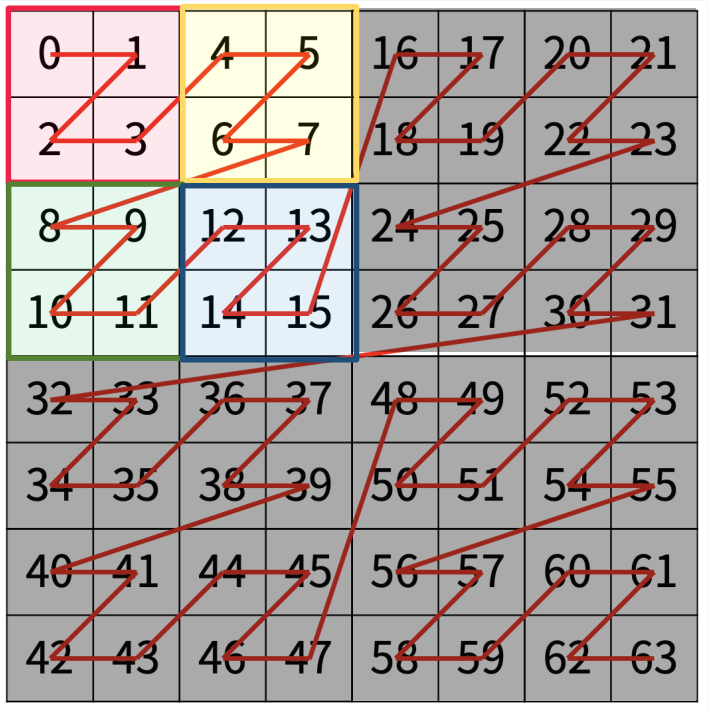
- 주어진 2차원 배열을 (n/2, n/2) 좌표를 기준으로 면적을 4등분 해보자
  - 각 구간을 (좌|상단) → (우|상단) → (좌|하단) → (우|하단) 순서로 탐색한다



BOJ1074: Z

문제 접근

- 주어진 2차원 배열을 (n/2, n/2) 좌표를 기준으로 면적을 4등분 해보자
  - 각 구간을 (좌|상단) → (우|상단) → (좌|하단) → (우|하단) 순서로 탐색한다



BOJ1074: Z

문제 접근

- 주어진 2차원 배열을 (n/2, n/2) 좌표를 기준으로 면적을 4등분 해보자
  - 각 구간도 (좌|상단) → (우|상단) → (좌|하단) → (우|하단) 순서로 탐색한다

→ 영역을 매 재귀마다 n/2 사이즈로 축소하여 판단할 수 있다

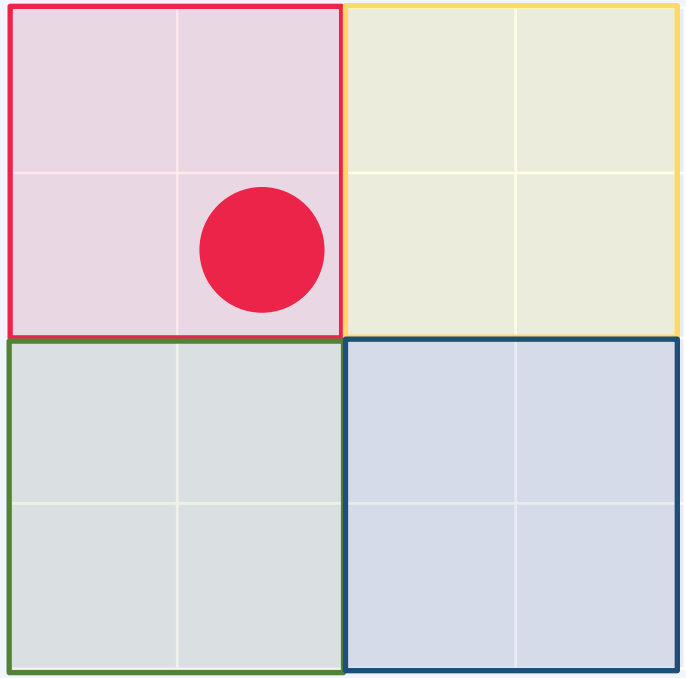
BOJ1074: Z

문제 접근
- (r, c)가 어느 영역에 있는지?

N == 2

4 * 4

mid = 2



r < mid
c < mid

좌 | 상단

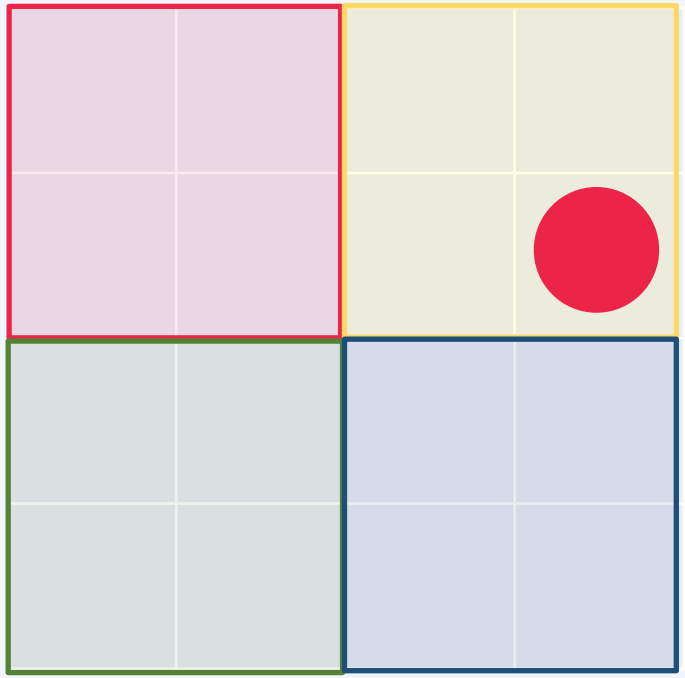
BOJ1074: Z

문제 접근
- (r, c)가 어느 영역에 있는지?

N == 2

4 * 4

mid = 2



r < mid
c >= mid

우 | 상단

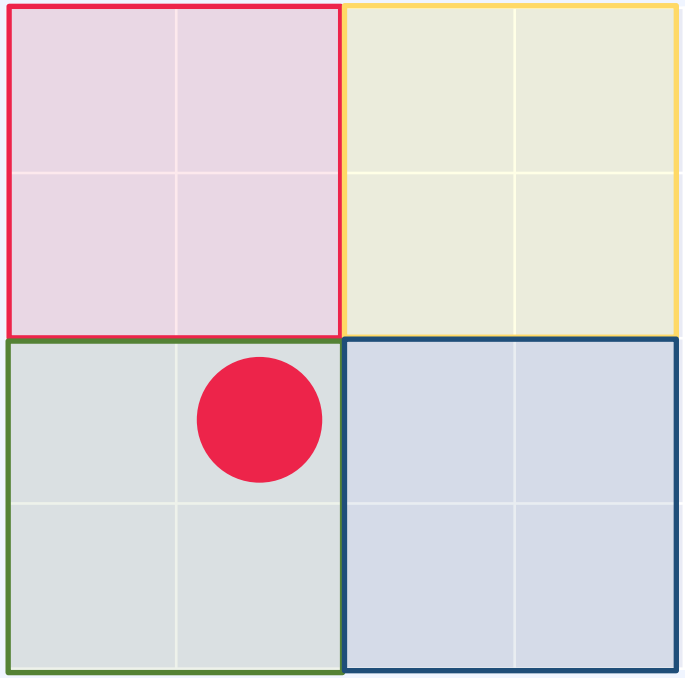
BOJ1074: Z

문제 접근
- (r, c)가 어느 영역에 있는지?

N == 2

4 * 4

mid = 2



r >= mid
c < mid

좌 | 하단

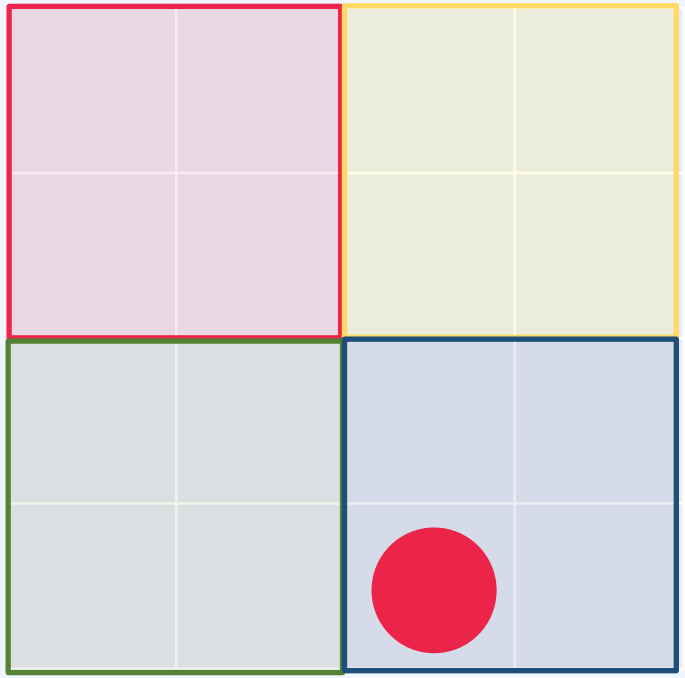
BOJ1074: Z

문제 접근
- (r, c)가 어느 영역에 있는지?

N == 2

4 * 4

mid = 2



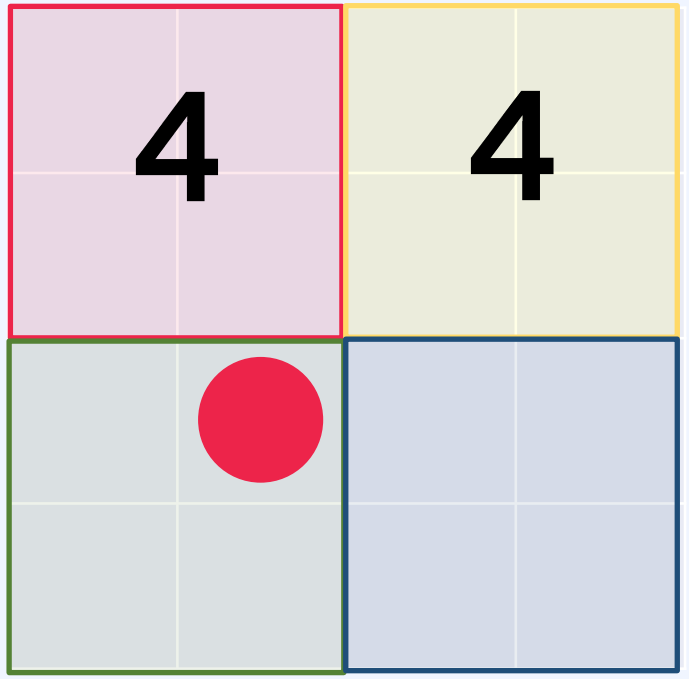
r >= mid
c >= mid

우 | 하단

BOJ1074: Z

문제 접근

- (r, c)가 어느 영역에 있는지?



(r, c)이 있는 영역을 이용해
계산 결과를 생략할 수 있다

좌|상단: {자신의 위치}
우|상단: 4 + {자신의 위치}
좌|하단: 4 + 4 + {자신의 위치}
우|하단: 4 + 4 + 4 + {자신의 위치}

→ 지나온 영역을 2^(n/2)로 한번에 더할 수 있다

BOJ1074: Z

문제 접근

1. 주어진 2차원 배열을 (n/2, n/2) 좌표를 기준으로 면적을 4등분 해보자
   - 각 구간을 (좌|상단) → (우|상단) → (좌|하단) → (우|하단) 순서로 탐색한다
   - 영역을 매 재귀마다 n/2 사이즈로 축소하여 판단할 수 있다

2. (r, c)가 어느 영역에 있는지?
   - 위치해 있는 영역을 파악하면 지나온 영역을 2^(n/2)로 한번에 더할 수 있다

BOJ1074: Z

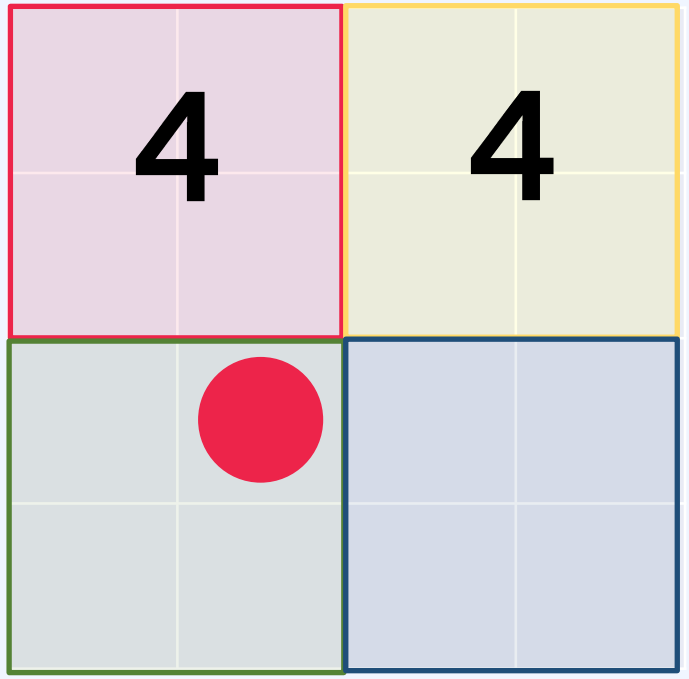
문제 접근

1. 주어진 2차원 배열을 (n/2, n/2) 좌표를 기준으로 면적을 4등분 해보자
   - 각 구간도 (좌|상단) → (우|상단) → (좌|하단) → (우|하단) 순서로 탐색한다
   - 영역을 매 재귀마다 n/2 사이즈로 축소하여 판단할 수 있다

```
public static void solve(int n, int r, int c) {
    int boardSize = 1 << n; // 2^n
    int mid = boardSize / 2;

    if (r < mid && c < mid)
        solve(n - 1, r, c);

    else if (r < mid && c >= mid)
        solve(n - 1, r, c - mid);

    else if (r >= mid && c < mid)
        solve(n - 1, r - mid, c);
    else
        solve(n - 1, r - mid, c - mid);
}
```

BOJ1074: Z

문제 접근
- (r, c)가 어느 영역에 있는지?



(r, c)이 있는 영역을 이용해
계산 결과를 생략할 수 있다

좌|상단: {자신의 위치}
우|상단: 4 + {자신의 위치}
좌|하단: 4 + 4 + {자신의 위치}
우|하단: 4 + 4 + 4 + {자신의 위치}

→ 지나온 영역을 2^(n/2)로 한번에 더할 수 있다

BOJ1074: Z

문제 접근

- (r, c)가 어느 영역에 있는지?

- 좌|상단: {자신의 위치}
- 우|상단: 2^(n/2) + {자신의 위치}
- 좌|하단: 2^(n/2) + 2^(n/2) + {자신의 위치}
- 우|하단: 2^(n/2) + 2^(n/2) + 2^(n/2) + {자신의 위치}

BOJ1074: Z

문제 접근

- (r, c)가 어느 영역에 있는지?

- 좌|상단: solve(n-1, r, c)
- 우|상단: 2^(mid * mid) + solve(n-1, r, c - mid)
- 좌|하단: 2^(mid * mid) + 2^(mid * mid) + solve(n-1, r – mid, c)
- 우|하단: 2^(mid * mid) + 2^(mid * mid) + 2^(mid * mid) + solve(n-1, r - mid, c - mid)

BOJ1074: Z

문제 접근

- (r, c)가 어느 영역에 있는지?

```
// 좌상단
if (r < mid && c < mid) {
    solve(n - 1, r, c);
}
// 우상단
else if (r < mid && c >= mid) {
    count += mid * mid;
    solve(n - 1, r, c - mid);
}
```

```
// 좌하단
else if (r >= mid && c < mid) {
    count += mid * mid * 2;
    solve(n - 1, r - mid, c);
}
// 우하단
else {
    count += mid * mid * 3;
    solve(n - 1, r - mid, c - mid);
}
```

Ch07. 분할 정복

3. [1992] 쿼드트리

BOJ1992: 쿼드트리

문제 요약

- 배열을 4등분하여 영역 구분
- 하나의 영역에 모두 같은 숫자로 구성되어 있다면?
  - 1개의 숫자로 표현하여 출력
- 다른 값이 섞여 있다면?
  - 해당 영역을 다시 4등분하여 위와 같이 반복
- 1 <= N <= 64

BOJ1992: 쿼드트리

문제 분석

- 영역을 4등분
  - 분할정복을 이용해 매 재귀마다 n/2 로 축소

- 영역의 값이 모두 동일한지 검사
  - 반복문 순회

BOJ1992: 쿼드 트리

ex) N == 4 인 흑백 영상

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

영역을 4등분 했을 때 시작 좌표
(mid == 4/2 == 2)

- 좌|상단: (0, 0)
- 우|상단: (0, 0 + mid)
- 좌|하단: (0 + mid, 0)
- 우|하단: (0 + mid, 0 + mid)

BOJ1992: 쿼드 트리

ex) N == 4 인 흑백 영상

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

같은 값으로 채워진 영역
- 좌|상단
- 우|하단

다른 값이 섞여서 채워진 영역
- 우|상단
- 좌|하단

BOJ1992: 쿼드 트리

ex) N == 4 인 흑백 영상

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

다른 값이 섞여서 채워진 영역
- 우|상단

→ 해당 영역에서 다시 4등분

BOJ1992: 쿼드 트리

ex) N == 4 인 흑백 영상

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

다른 값이 섞여서 채워진 영역
- 우|상단

→ 해당 영역에서 다시 4등분

BOJ1992: 쿼드 트리

ex) N == 4 인 흑백 영상

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

영역을 4등분 했을 때 시작 좌표
(mid == 2/2 == 1)

- 좌|상단: (0, 3)
- 우|상단: (0, 3 + mid)
- 좌|하단: (0 + mid, 3)
- 우|하단: (0 + mid, 3 + mid)

BOJ1992: 쿼드 트리

데이터 입력

```
for(int i=0; i<n; i++) {
    String str = sc.next();
    for(int j=0; j<n; j++) {
        board[i][j] = str.charAt(j) - '0';
    }
}
```

입력이 공백 없이 주어진다
라인 단위로 입력을 String으로 받고, 아스키 코드 '0'을 빼서 숫자 배열로 처리한다

BOJ1992: 쿼드 트리

분할

영역을 4등분 했을 때 시작 좌표
(mid == length / 2)

- 좌|상단: (r, c)
- 우|상단: (r, c + mid)
- 좌|하단: (r + mid, c)
- 우|하단: (r + mid, c + mid)

```
public static void compress(int r, int c, int length) {
    int mid = length / 2;
    compress(r, c, mid);
    compress(r, c + mid, mid);
    compress(r + mid, c, mid);
    compress(r + mid, c + mid, mid);
}
```

BOJ1992: 쿼드 트리

정복 / 조합

영역이 모두 같은 숫자인지 검사

- (r, c) 시작점과 대조
- length 길이만큼 반복문으로 검사

```
public static boolean isSame(int r, int c, int length) {
    for(int i = 0; i < length; i++) {
        for(int j = 0; j < length; j++) {
            if(board[r][c] != board[r+i][c+j]) return false;
        }
    }
    return true;
}
```

```
if(isSame(r, c, length)) {
    System.out.print(board[r][c]);
    return;
}
```

Ch07. 분할 정복

4. [1780] 종이의 개수

BOJ1780: 종이의 개수

문제 요약

- N * N 크기의 행렬로 표현되는 종이
  - 각 칸에는 -1, 0, 1 중 하나가 저장

- 종이가 모든 같은 수로 구성
  - 그대로 사용
- 종이의 숫자가 섞여 있는 경우
  - 9개의 영역 (3 * 3) 으로 나누고, 위의 과정을 반복

- 모든 영역에 대해서 만들어지는 종이의 개수 출력

- 1 <= N <= 3^7, N은 3^k 꼴

BOJ1780: 종이의 개수

입력 데이터
9 0 0 0 1 1 1 -1 -1 -1 0 0 0 1 1 1 -1 -1 -1 0 0 0 1 1 1 -1 -1 -1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 -1 0 1 -1 0 1 -1 0 -1 1 0 1 -1 0 1 -1 0 1 -1 1 0 -1 0 1 -1

출력 데이터
10 12 11

BOJ1780: 종이의 개수

입력 데이터

9
0 0 0 1 1 1 -1 -1 -1
0 0 0 1 1 1 -1 -1 -1
0 0 0 1 1 1 -1 -1 -1
1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0
1 1 1 0 0 0 0 0 0
0 1 -1 0 1 -1 0 1 -1
0 -1 1 0 1 -1 0 1 -1
0 1 -1 1 0 -1 0 1 -1

출력 데이터

10
12
11

BOJ1780: 종이의 개수

문제 분석

- (r, c) 를 기준으로 length*length 영역이 모두 같은 종이면?
  - 종이 카운트 증가
- 서로 다른 종이가 섞여 있다면?
  - 영역을 9분할하여 재귀함수 수행

BOJ1780: 종이의 개수

(r, c) 를 기준으로 length*length 영역이 모두 같은 종이면?

- 종이 카운트 증가

```
public static boolean isSame(int r, int c, int length) {
    for (int i = 0; i < length; i++) {
        for (int j = 0; j < length; j++) {
            if (board[r][c] != board[r + i][c + j]) return false;
        }
    }
    return true;
}
```

```
if (isSame(r, c, length)) {
    int color = board[r][c] + 1;
    paper[color]++;
    return;
}
```

BOJ1780: 종이의 개수

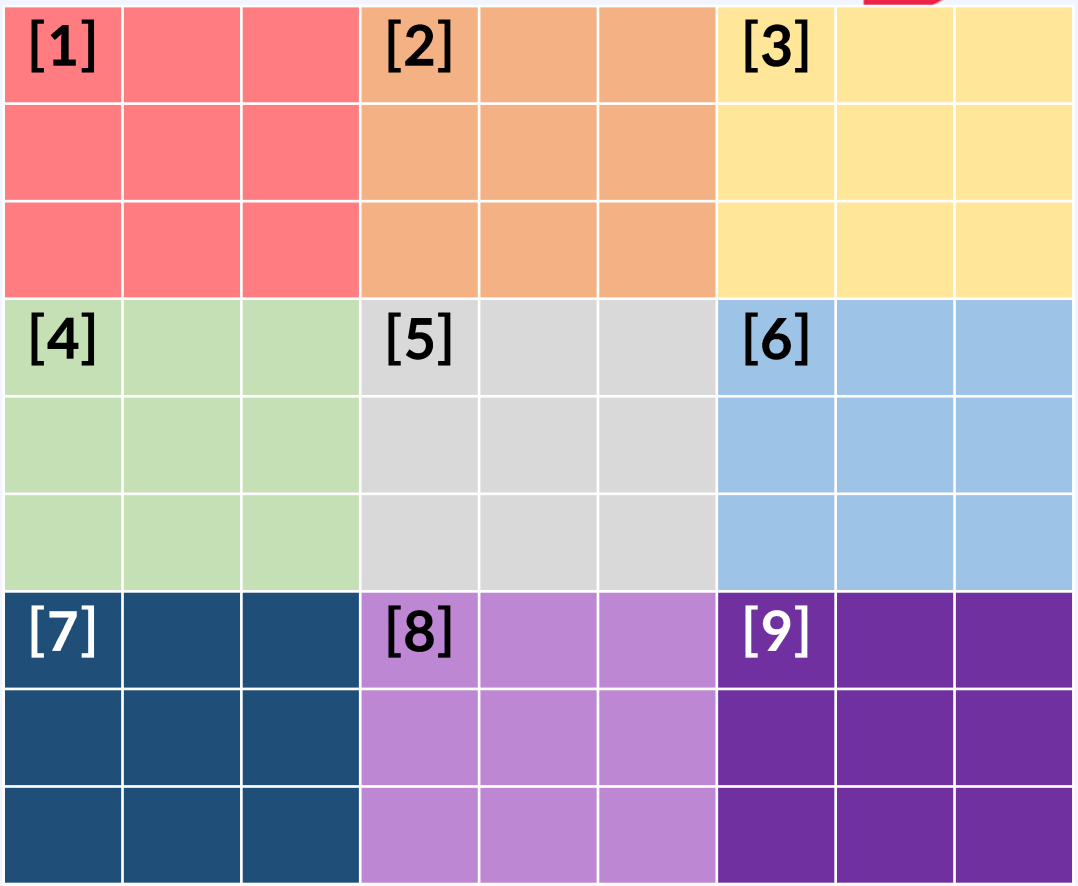
문제 분석

- N * N 크기의 행렬을 9분할 (행 / 3, 열 / 3)
- 행렬의 좌측 상단의 좌표를 (r, c), 행렬의 행 길이가 length 일 때 9분할된 각 영역의 시작 좌표는?
  - (r, c)
  - (r, c + length / 3)
  - (r, c + length / 3 + length / 3)
  - (r + length / 3, c)
  - (r + length / 3, c + length / 3)
  - (r + length / 3, c + length / 3 + length / 3)
  - (r + length / 3 + length / 3, c)
  - (r + length / 3 + length / 3, c + length / 3)
  - (r + length / 3 + length / 3, c + length / 3 + length / 3)

BOJ1780: 종이의 개수

행렬의 좌측 상단의 좌표를 (r, c), 행렬의 행 길이가 length 일 때

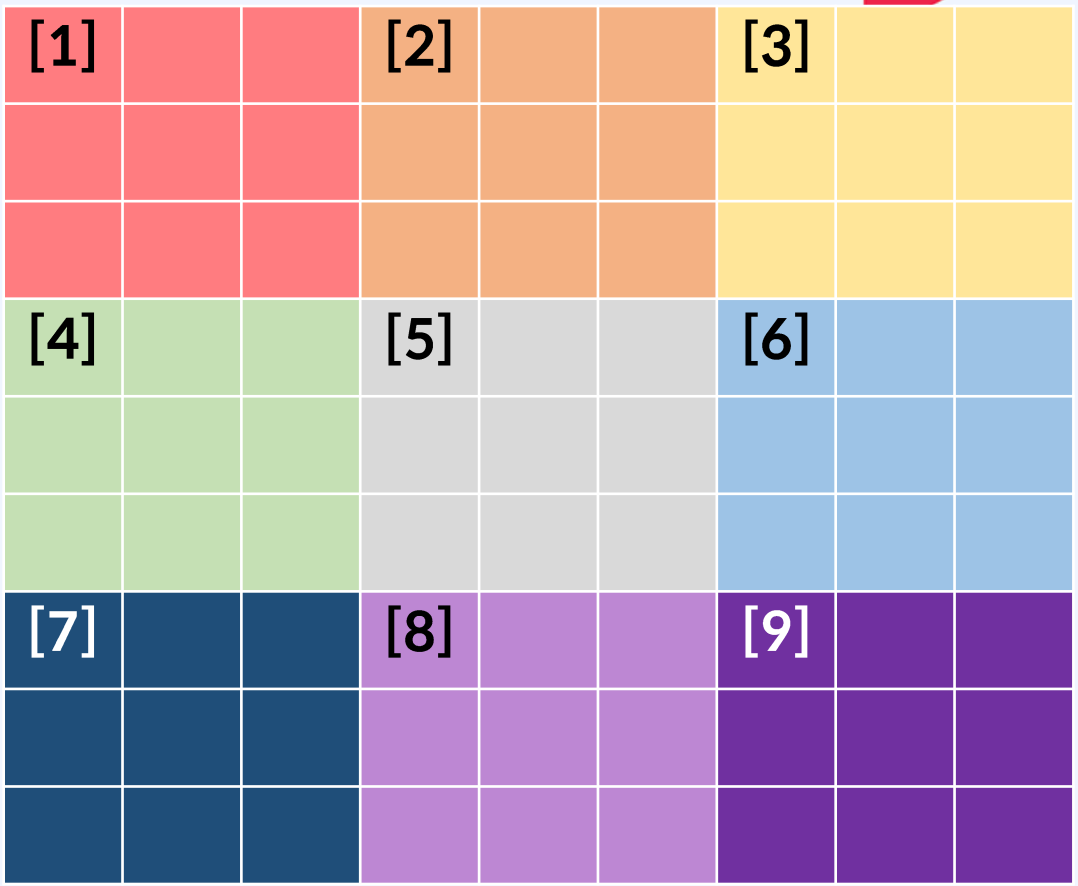
9분할된 각 영역의 시작 좌표는?



- [1] (r, c)
- [2] (r, c + length / 3)
- [3] (r, c + length / 3 + length / 3)
- [4] (r + length / 3, c)
- [5] (r + length / 3, c + length / 3)
- [6] (r + length / 3, c + length / 3 + length / 3)
- [7] (r + length / 3 + length / 3, c)
- [8] (r + length / 3 + length / 3, c + length / 3)
- [9] (r + length / 3 + length / 3, c + length / 3 + length / 3)

BOJ1780: 종이의 개수

```
int next = length / 3;
for (int r = 0; r < 3; r++) {
    for (int c = 0; c < 3; c++) {
        cut(row + r * next, col + c * next, next);
    }
}
```



- [1] (r, c)
- [2] (r, c + length / 3)
- [3] (r, c + length / 3 + length / 3)
- [4] (r + length / 3, c)
- [5] (r + length / 3, c + length / 3)
- [6] (r + length / 3, c + length / 3 + length / 3)
- [7] (r + length / 3 + length / 3, c)
- [8] (r + length / 3 + length / 3, c + length / 3)
- [9] (r + length / 3 + length / 3, c + length / 3 + length / 3)

Ch07. 분할 정복

5. [2751] 수 정렬하기 2

BOJ2751: 수 정렬하기 2

문제 요약

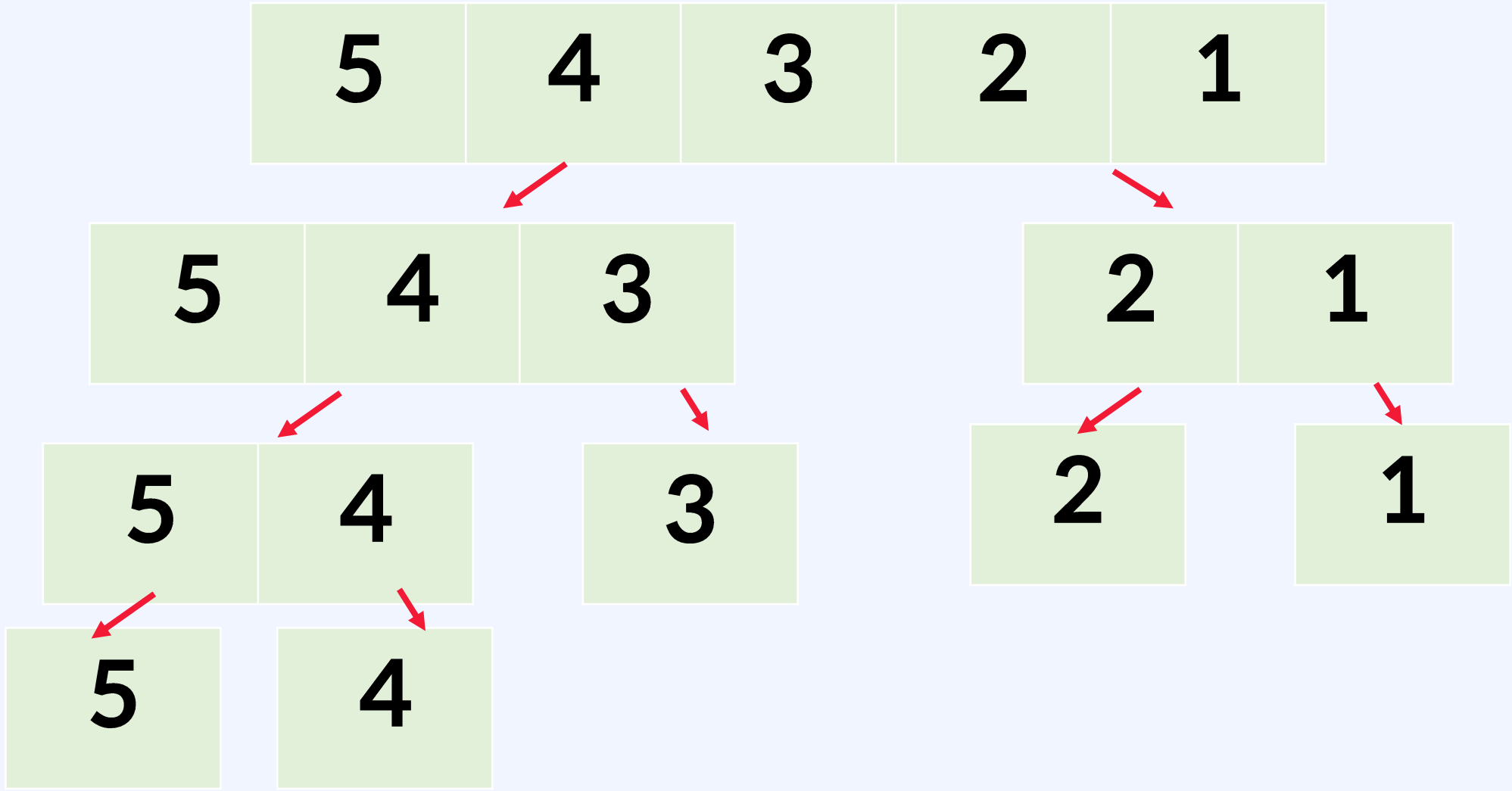
- N개의 수가 입력으로 주어짐
- 오름차순 정렬
- 1 <= N <= 1,000,000

BOJ2751: 수 정렬하기 2

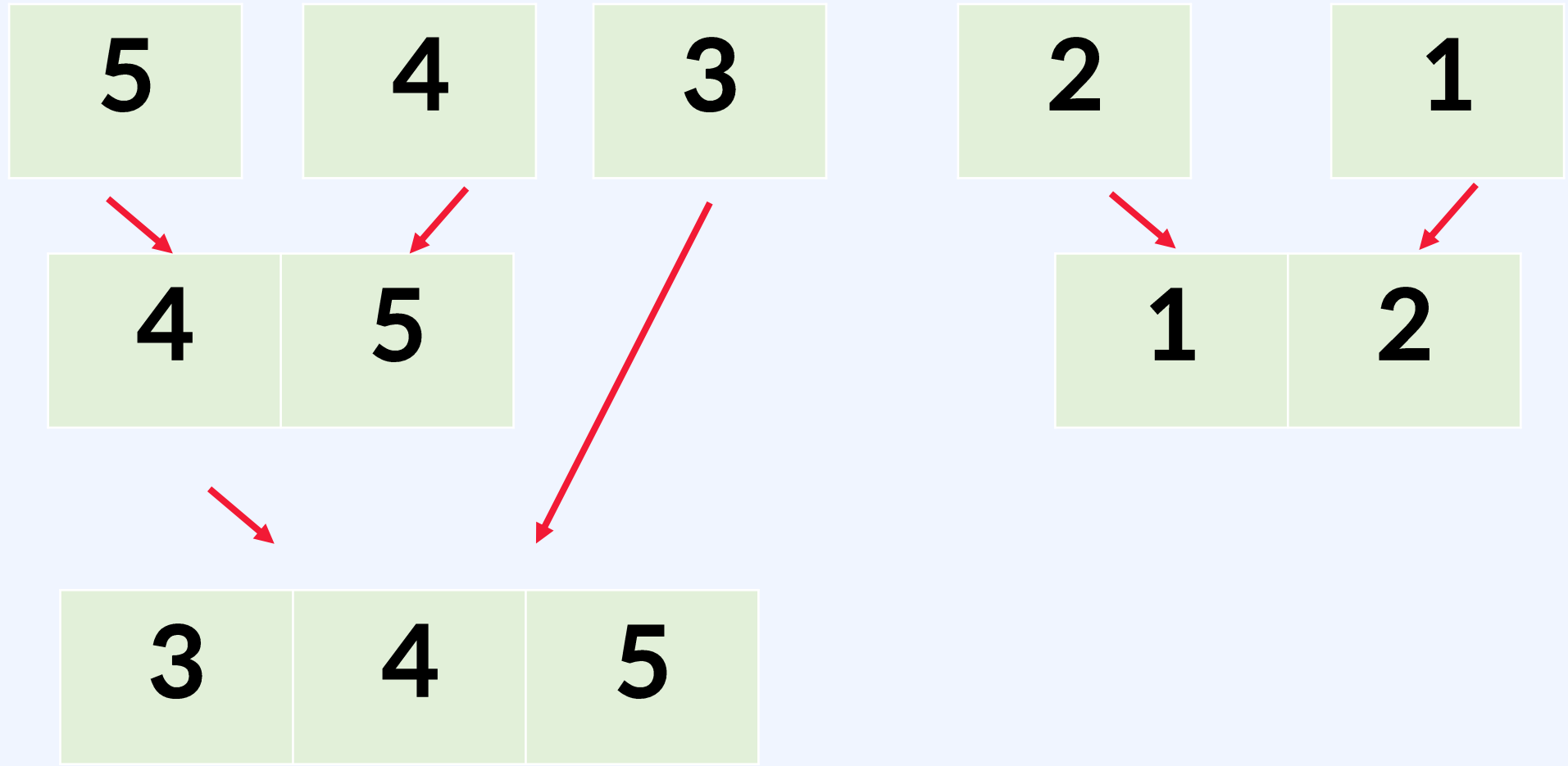
문제 분석

- N의 사이즈가 1백만으로
  O(N^2) 시간 복잡도의 정렬을 사용하면 시간 초과가 발생함
- O(N log N) 속도의 정렬을 수행해야 함
- 분할정복 챕터이므로, 합병 정렬을 이용해 구현

BOJ2751: 수 정렬하기 2



BOJ2751: 수 정렬하기 2

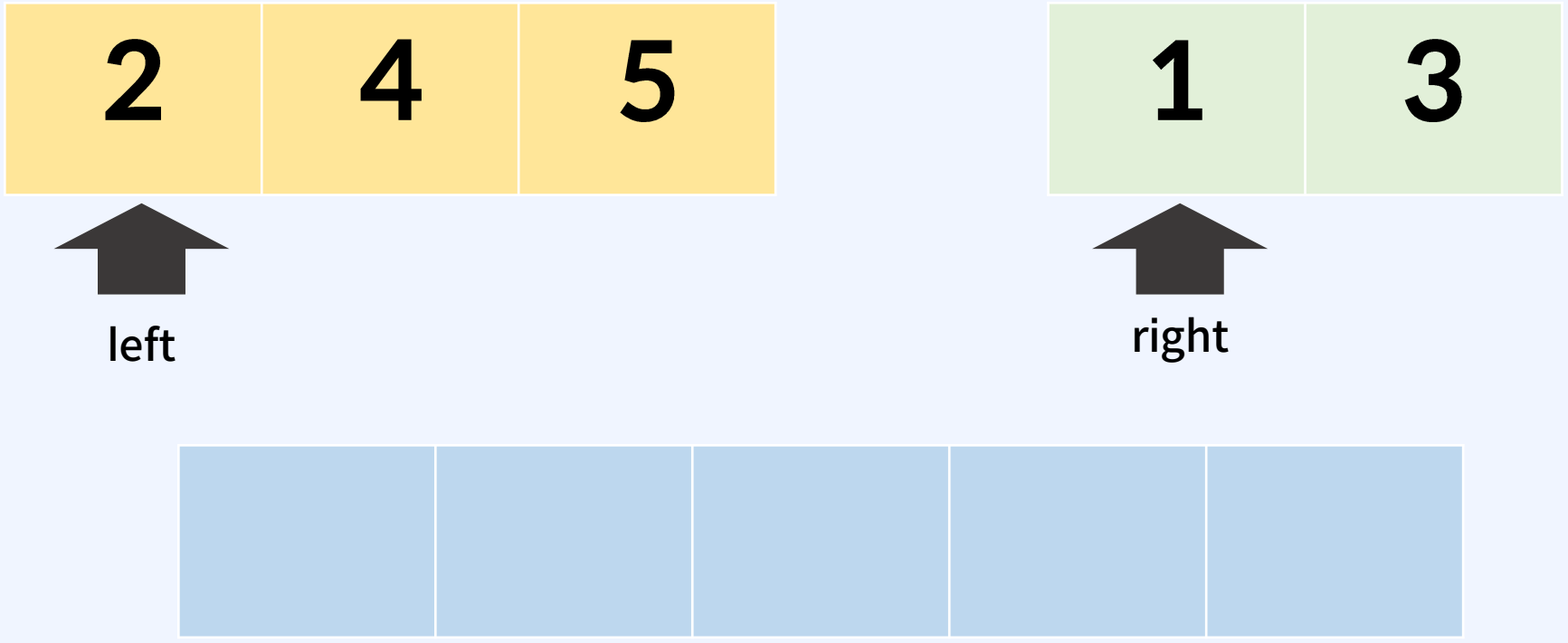


BOJ2751: 수 정렬하기 2

문제를 분할하고, 하나로 합치는 과정에서 오름차순으로 모은다

BOJ2751: 수 정렬하기 2

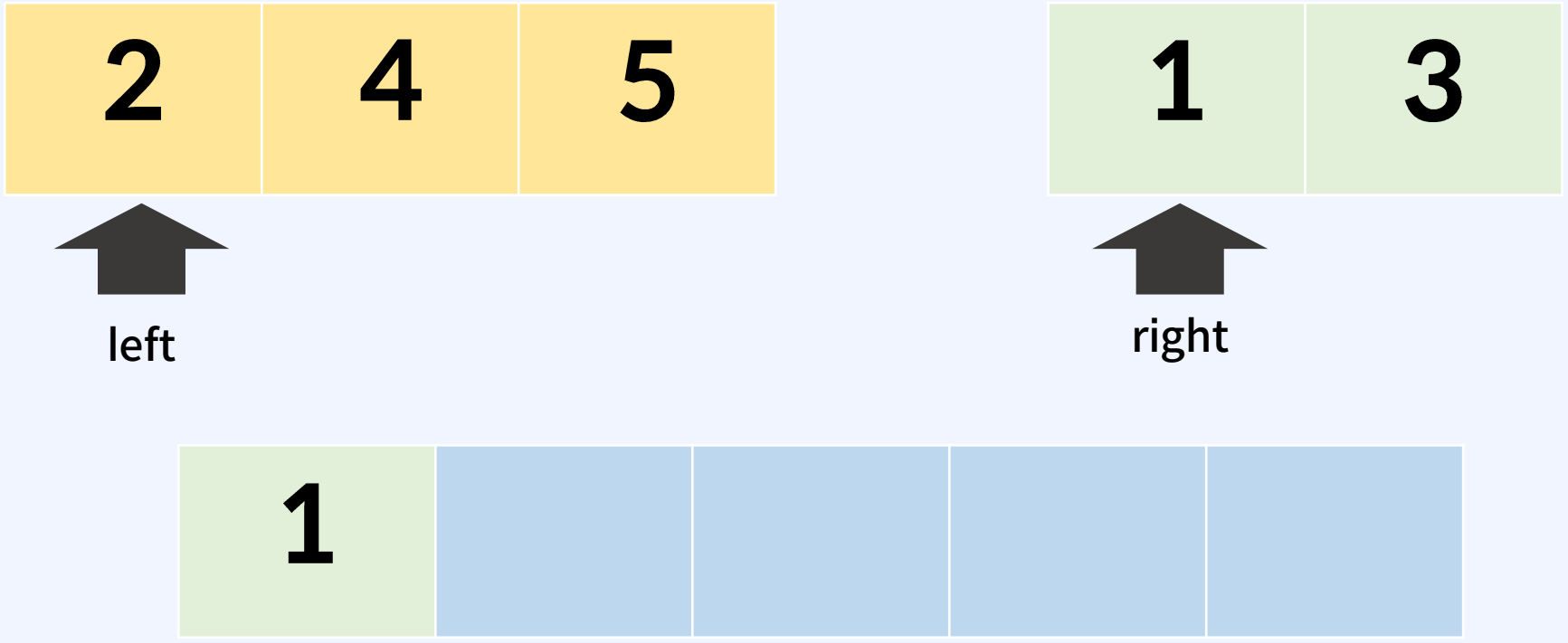
합치기?



left, right 중에 작은 수를 먼저 고른다

BOJ2751: 수 정렬하기 2

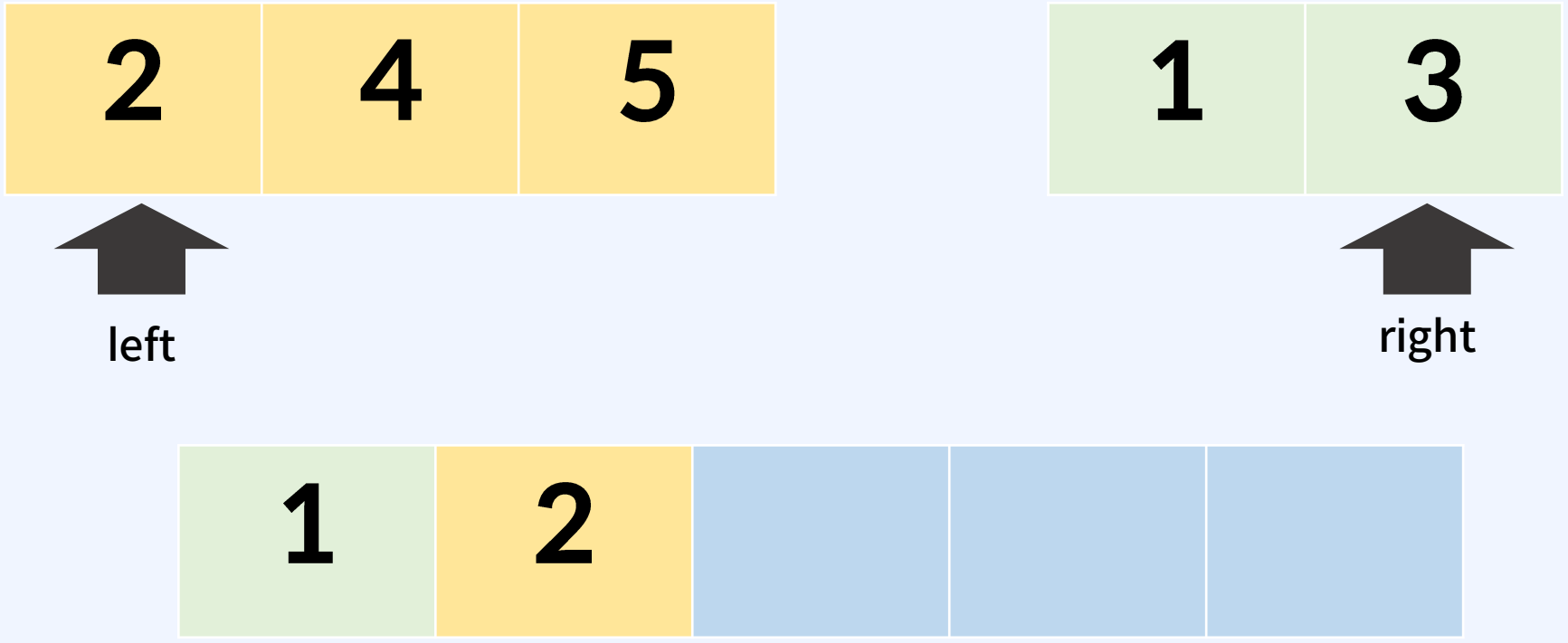
합치기?



left, right 중에 작은 수를 먼저 고른다

BOJ2751: 수 정렬하기 2

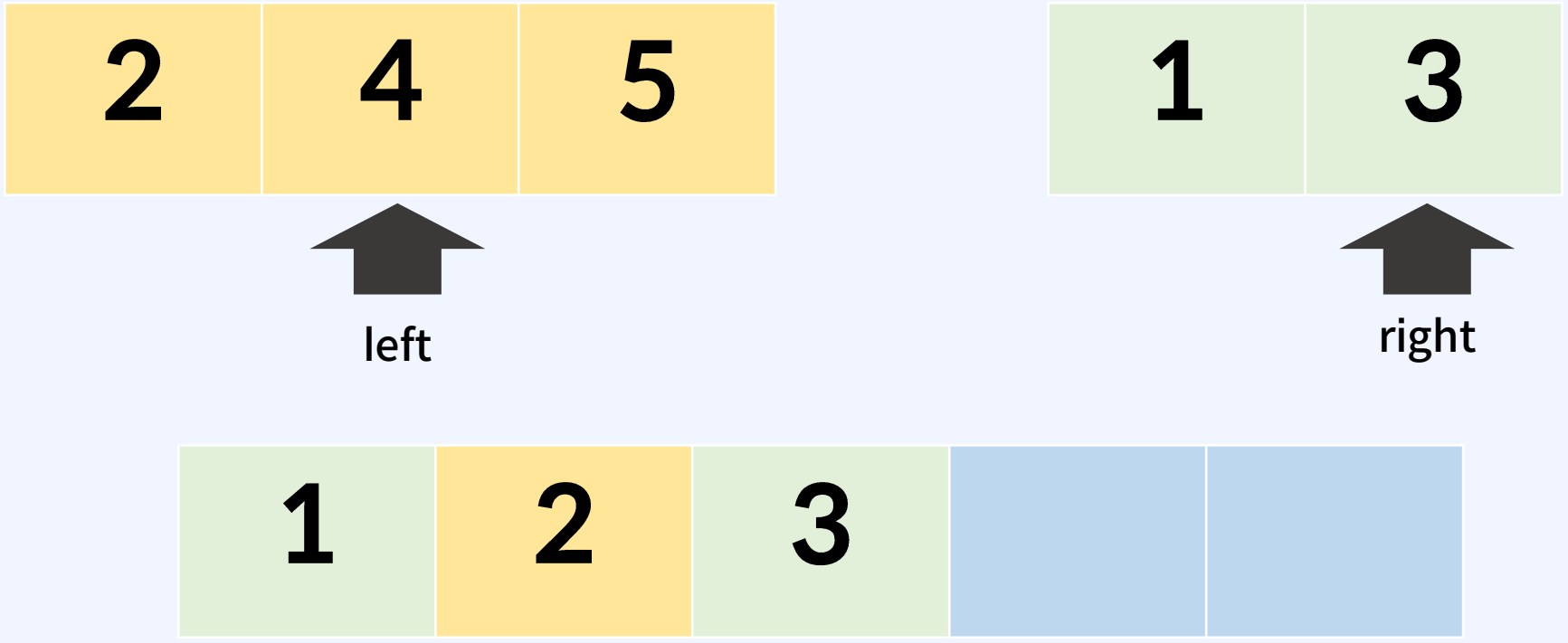
합치기?



left, right 중에 작은 수를 먼저 고른다

BOJ2751: 수 정렬하기 2

합치기?



left, right 중에 작은 수를 먼저 고른다

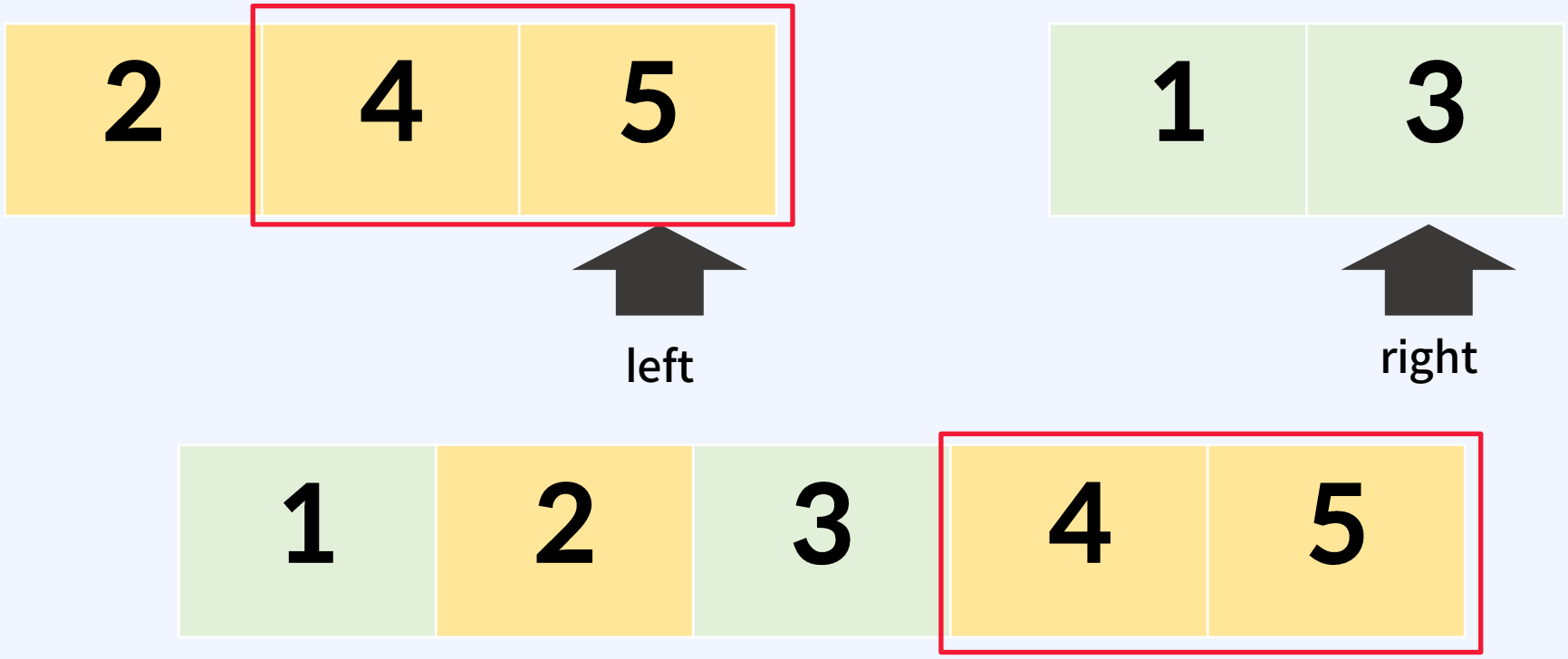
BOJ2751: 수 정렬하기 2

합치기?

right 가 끝에 도달하면, left에 남은 원소를 그대로 넣는다
(반대도 성립)

BOJ2751: 수 정렬하기 2

합치기?



right 가 끝에 도달하면, left에 남은 원소를 그대로 넣는다
(반대도 성립)

BOJ2751: 수 정렬하기 2

분할

```
public static void divide(int start, int end) {
    if (start == end) return;
    int mid = (start + end) / 2;
    divide(start, mid);
    divide(mid + 1, end);
    combine(start, end);
}
```

왼쪽 (start, mid) / 오른쪽 (mid + 1, end) 영역으로 나누고
합치면서 오름차순 정렬을 수행한다

BOJ2751: 수 정렬하기 2

정복, 조합

왼쪽 / 오른쪽 수 중에
더 작은 값을 골라

temp[newIdx++] 에
배치한다

```
static void combine(int start, int end) {
    int mid = (start + end) / 2;
    int left = start;
    int right = mid + 1;
    int newIdx = start;

    while (left <= mid && right <= end) {
        if (numbers[left] < numbers[right]) {
            temp[newIdx++] = numbers[left++];
        } else {
            temp[newIdx++] = numbers[right++];
        }
    }
```

BOJ2751: 수 정렬하기 2

정복, 조합

왼쪽 혹은 오른쪽 영역에
수가 남아 있다면

남은 수열을
그대로 배치한다

```
while (left <= mid) {
    temp[newIdx++] = numbers[left++];
}

while (right <= end) {
    temp[newIdx++] = numbers[right++];
}
```