

# 분할 정보

최백준 [choi@startlink.io](mailto:choi@startlink.io)

---

# 분할 정보

---

# 분할 정복

Divide & Conquer

3

- 분할 정복은 문제를 2개 또는 그 이상의 작은 부분 문제로 나눈 다음 푸는 것(분할)
- 푼 다음에는 다시 합쳐서 정답을 구할 때도 있음 (정복)
- 대표적인 분할 정복 알고리즘
- 퀵 소트
- 머지 소트
- 큰 수 곱셈 (카라추바 알고리즘)
- FFT

# 분할 정복

Divide & Conquer

- 분할 정복과 다이나믹은
- 문제를 작은 부분 문제로 나눈다는 점은 동일하다
- 분할 정복: 문제가 겹치지 않음
- 다이나믹: 문제가 겹쳐서 겹치는 것을 Memoization으로 해결

# 이분 탐색

---

# 이분 탐색

Binary Search

- 정렬되어 있는 리스트에서 어떤 값을 빠르게 찾는 알고리즘
- 리스트의 크기를  $N$  이라고 했을 때
- $\lg N$ 의 시간이 걸린다.

# 이분 탐색

Binary Search

- 4를 찾아보자
- $L = 0, R = 8, M = 4$
- $4 < 7$  이기 때문에 왼쪽 ( $L \sim M-1$ )에 4가 있을 수 있다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



4

$O(N)$

$(\lg N)$

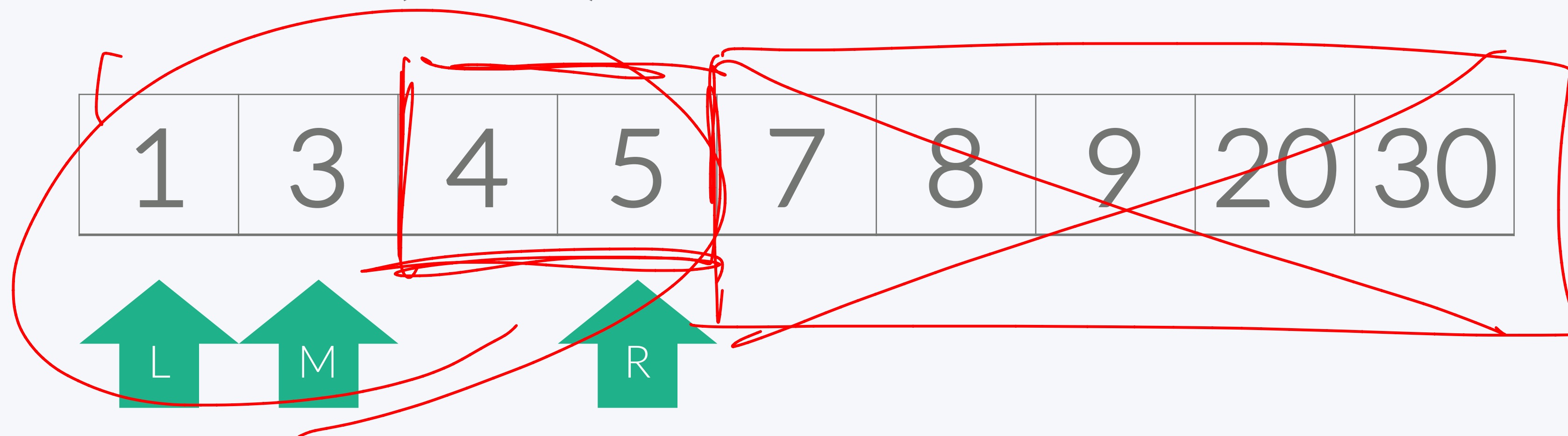
$k = \lg x$

$2^k = x$

# 이분 탐색

## Binary Search

- 4를 찾아보자
- $L = 0, R = 3, M = 1$
- $4 > 3$  이기 때문에 오른쪽 ( $M+1 \sim R$ )에 4가 있을 수 있다.





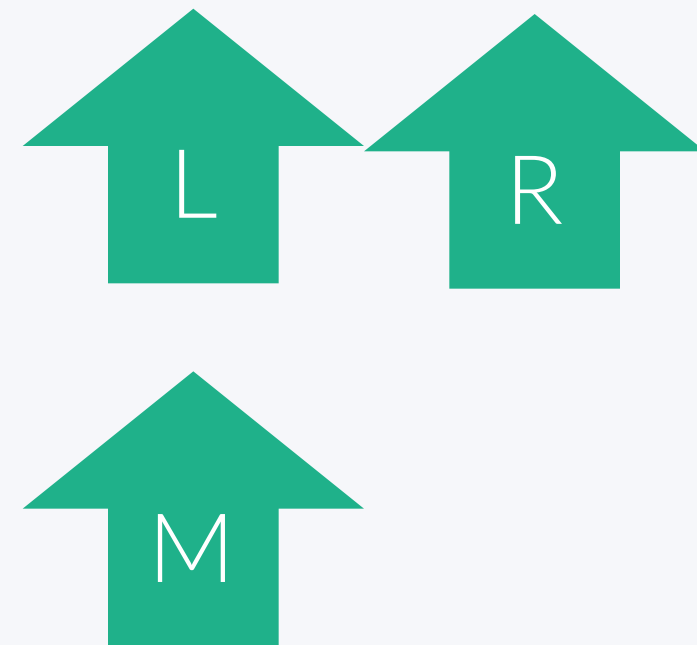
# 이분 탐색

## Binary Search

9

- 4를 찾아보자
- $L = 2, R = 3, M = 2$
- $4 == 4$  이다. 4를 찾았다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



# 이분 탐색

10

## Binary Search

- 2를 찾아보자
- $L = 0, R = 8, M = 4$
- $2 < 7$  이기 때문에 왼쪽 ( $L \sim M-1$ )에 4가 있을 수 있다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



# 이분 탐색

11

## Binary Search

- 2를 찾아보자
- $L = 0, R = 3, M = 1$
- $2 < 3$  이기 때문에 왼쪽 ( $L \sim M-1$ )에 2가 있을 수 있다.

1	3	4	5	7	8	9	20	30
 L		 M		 R				

# 이분 탐색

12

## Binary Search

- 2를 찾아보자
- $L = 0, R = 0, M = 0$
- $2 > 1$  이기 때문에 오른쪽 ( $M+1 \sim R$ )에 2가 있을 수 있다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



# 이분 탐색

## Binary Search

13

- 2를 찾아보자
- $L = 1, R = 0, M = 0$
- $L < R$  이기 때문에, 이분 탐색을 종료한다. 2는 리스트에 없다.

1	3	4	5	7	8	9	20	30
---	---	---	---	---	---	---	----	----



# 이분 탐색

## Binary Search

14

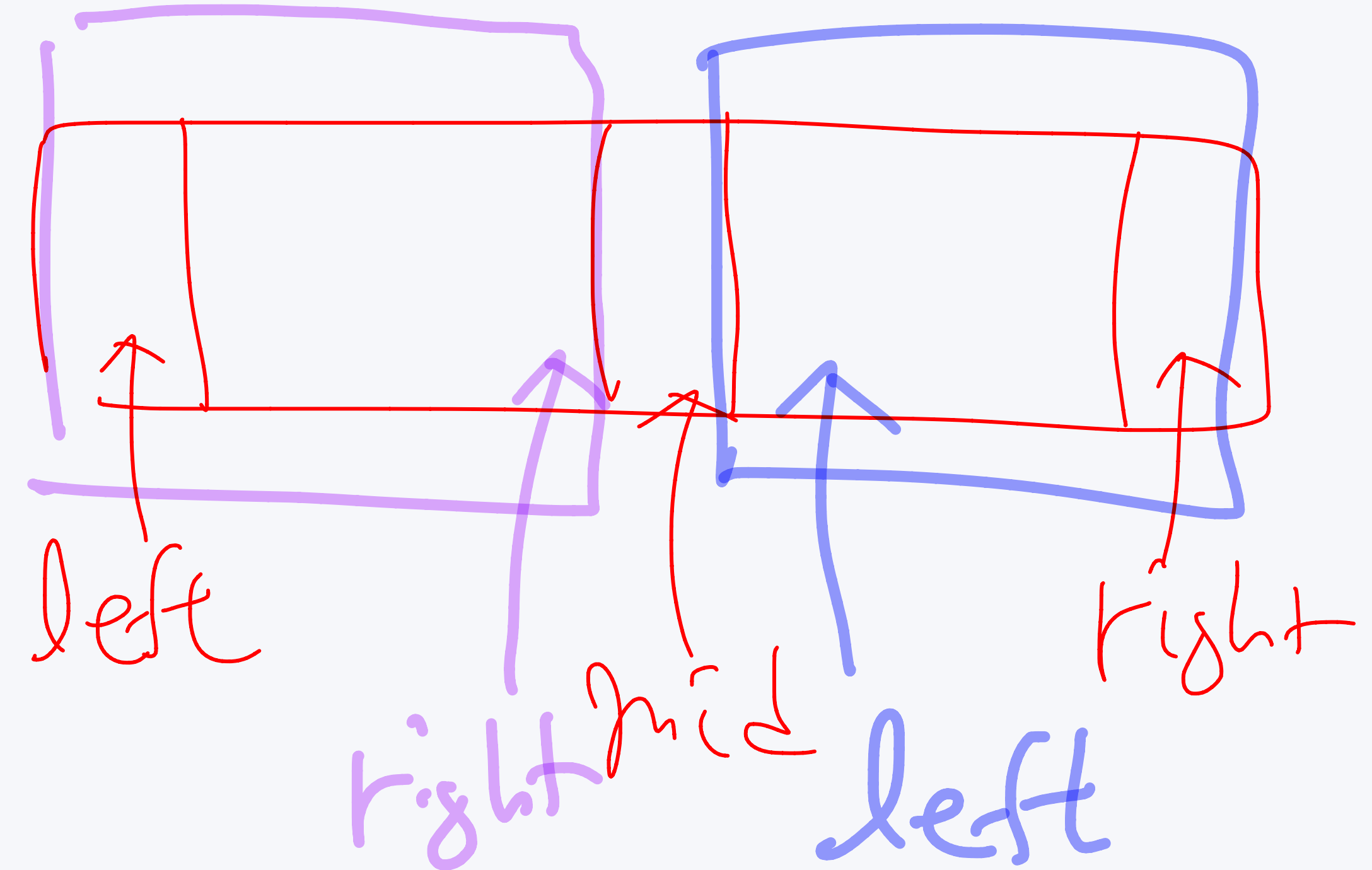
- 정렬되어 있는 리스트에서 어떤 값을 빠르게 찾는 알고리즘
- 리스트의 크기를  $N$  이라고 했을 때
- $\lg N$ 의 시간이 걸린다.
- 시간 복잡도가  $\lg N$ 인 이유는
- 크기가  $N$ 인 리스트를 계속해서 절반으로 나누기 때문이다.
- $2^k = N$  일 때,  $k = \lg N$

# 이분 탐색

Binary Search

```
while (left <= right) {  
    int mid = (left + right) / 2;  
    if (a[mid] == x) {  
        position = mid;  
        break;  
    } else if (a[mid] > x) {  
        right = mid - 1;  
    } else {  
        left = mid + 1;  
    }  
}
```

$$\textcircled{X} = a[\text{mid}]$$



$left < right$

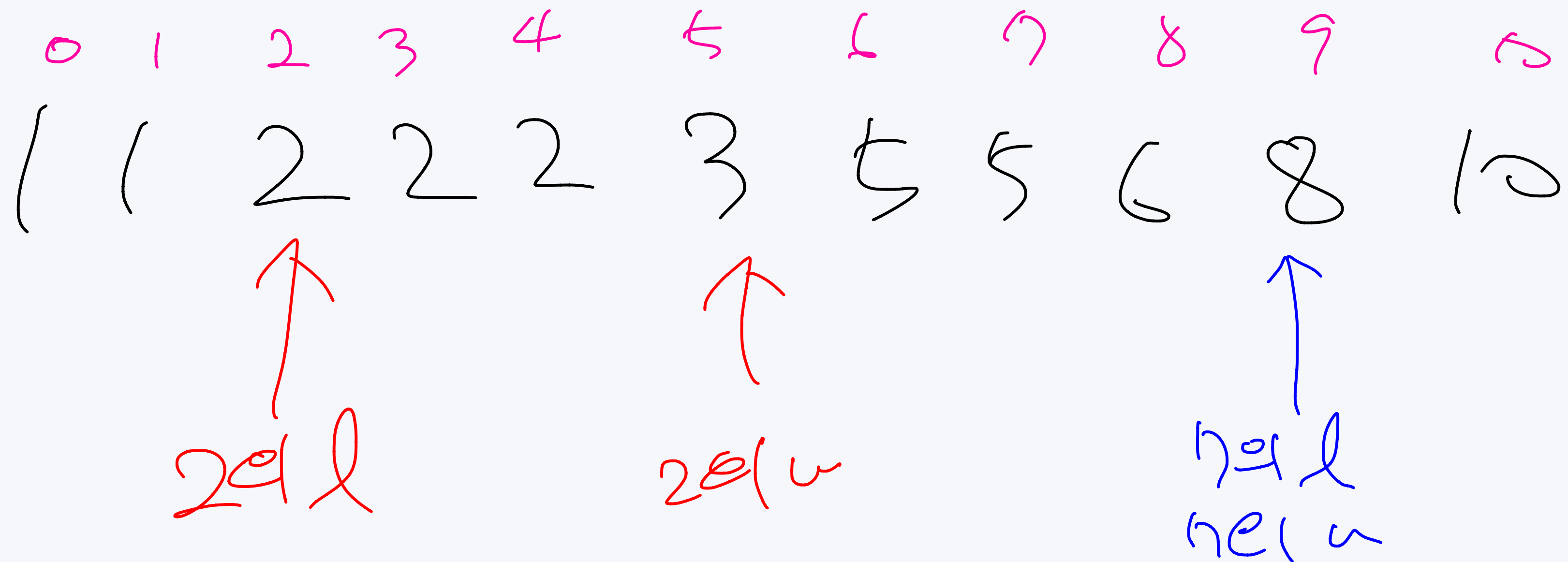
# 숫자 카드

16

<https://www.acmicpc.net/problem/10815>

- 이분 탐색을 이용해 풀 수 있다.

lower bound X보다 크거나 같으면서 가장 왼쪽에 있는 수  
upper bound X보다 크면서 가장 오른쪽에 있는 수





# 숫자 카드

<https://www.acmicpc.net/problem/10815>

- C++ (이분 탐색 구현): <https://gist.github.com/Baekjoon/bd0c441b31f85245a7ac>
- C++ (STL): <https://gist.github.com/Baekjoon/52f8f63c1863be93f7d1>
- Java: <https://gist.github.com/Baekjoon/5b2b2314e45b40116b88b25d4ed40c28>

# 숫자 카드 2

<https://www.acmicpc.net/problem/10816>

- 이분 탐색을 이용해 풀 수 있다.

# 숫자 카드 2

<https://www.acmicpc.net/problem/10816>

- C++ (equal\_range): <https://gist.github.com/Baekjoon/896c0b7c7bfb1c49bc08>
- C++ (lower\_bound, upper\_bound): <https://gist.github.com/Baekjoon/5d8e1fcdf1ed3f240201>
- C++ (multiset 사용): <https://gist.github.com/Baekjoon/98a2993e3d1258221f69>

$N \lg N$

# 머지 소트

---

# 머지 소트

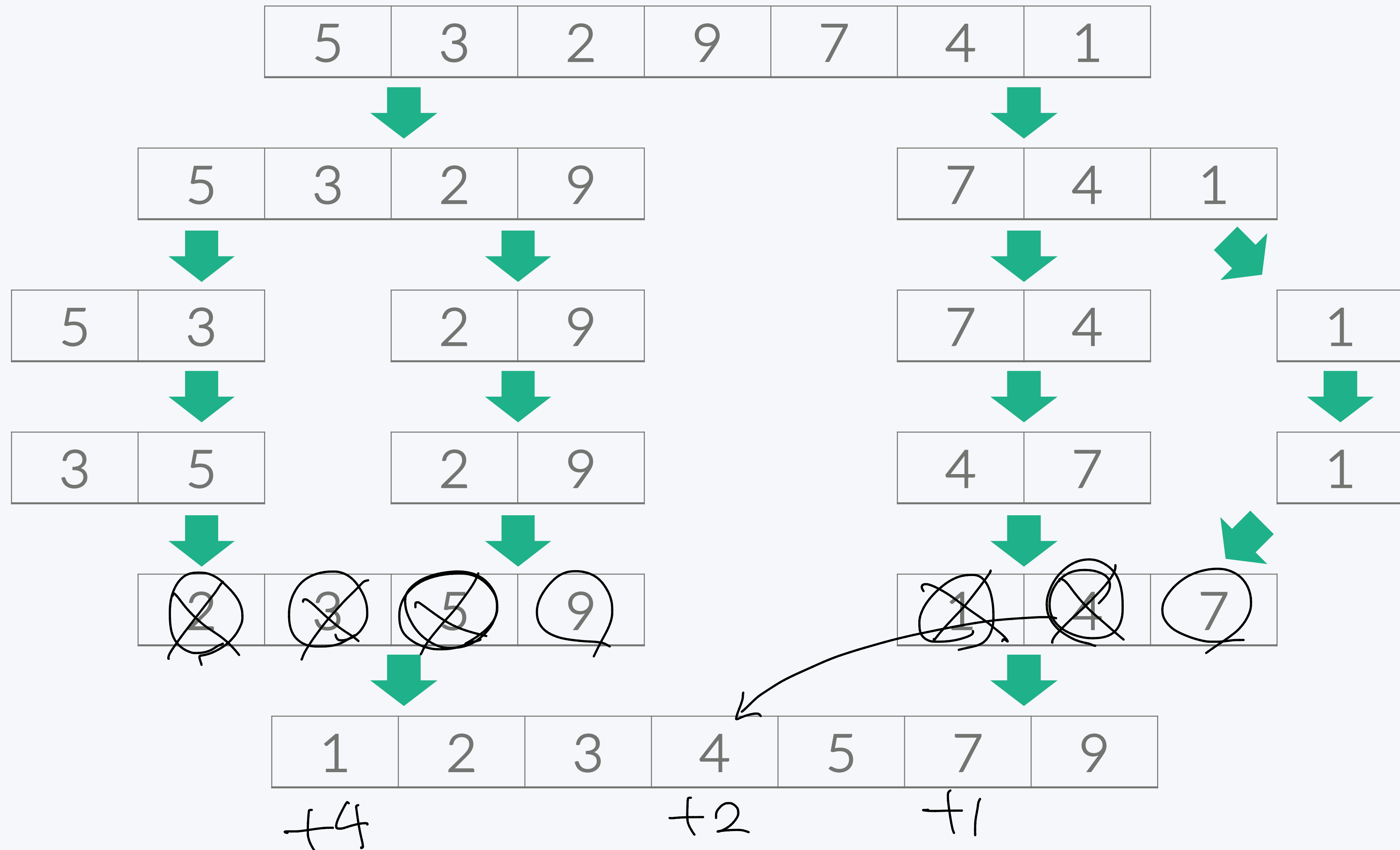
Merge Sort

- N개를 정렬하는 알고리즘
- N개를  $N/2$ ,  $N/2$ 개로 나눈다.
- 왼쪽  $N/2$ 와 오른쪽  $N/2$ 를 정렬한다.
- 두 정렬한 결과를 하나로 합친다.

# 머지 소트

Merge Sort

22



# 머지 소트

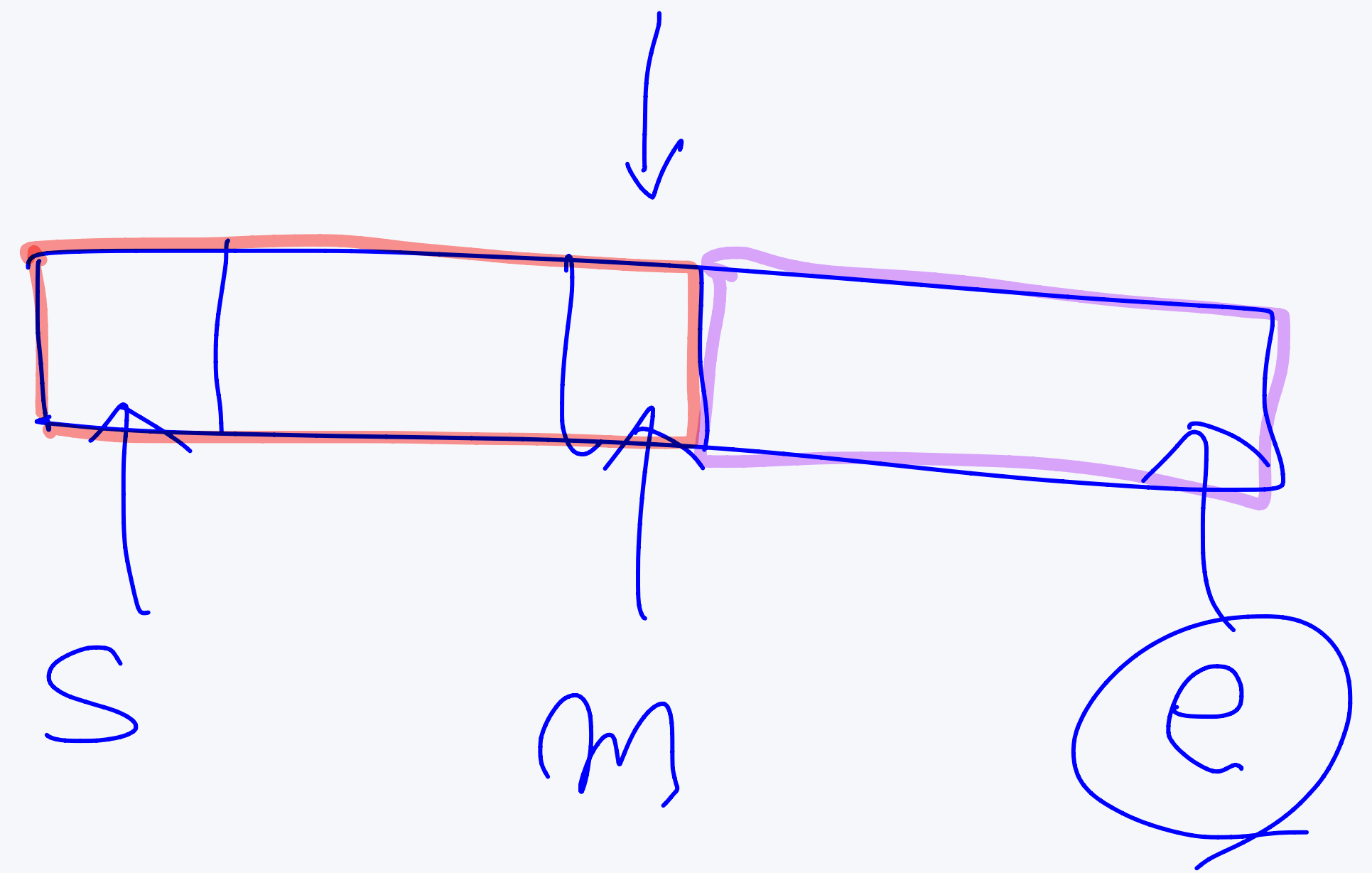
## Merge Sort

```
void sort(int start, int end) {  
    if (start == end) {  
        return;  
    }  
    int mid = (start+end)/2;  
    sort(start, mid);  
    sort(mid+1, end);  
    merge(start, end);  
}
```

# 머지 소트

## Merge Sort

```
void merge(int start, int end) {  
    int mid = (start+end)/2;  
    int i = start, j = mid+1, k = 0;  
    while (i <= mid && j <= end) {  
        if (a[i] <= a[j]) b[k++] = a[i++];  
        else b[k++] = a[j++];  
    }  
    while (i <= mid) b[k++] = a[i++];  
    while (j <= end) b[k++] = a[j++];  
    for (int i=start; i<=end; i++) {  
        a[i] = b[i-start];  
    }  
}
```



두 부분이 합쳐져  
정렬됨

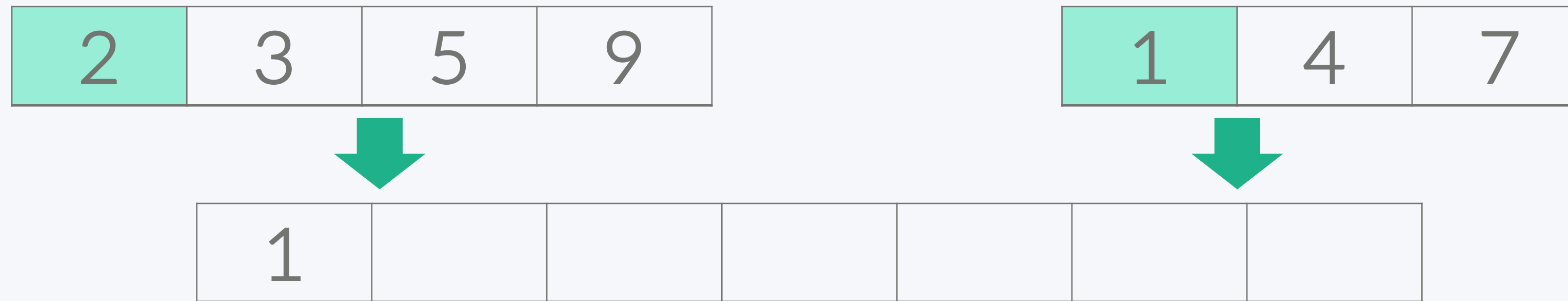


# 배열 합치기

25

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



# 배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현

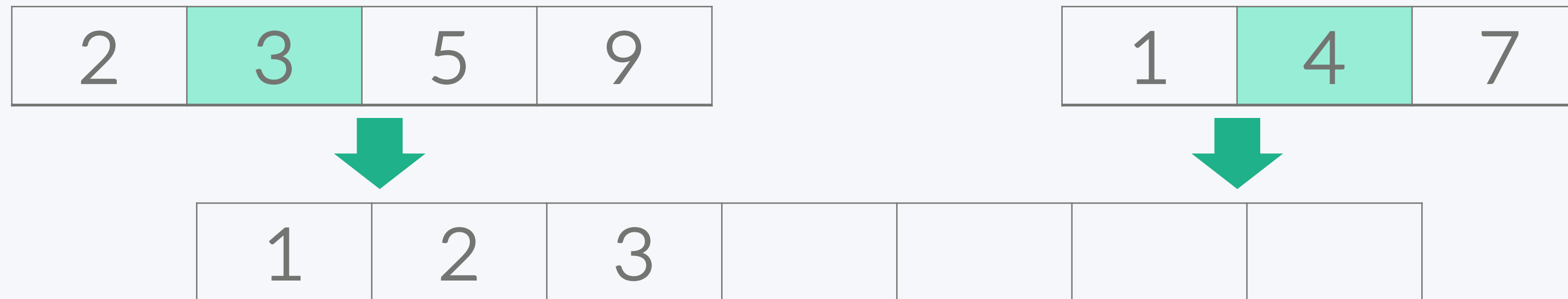


# 배열 합치기

27

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현

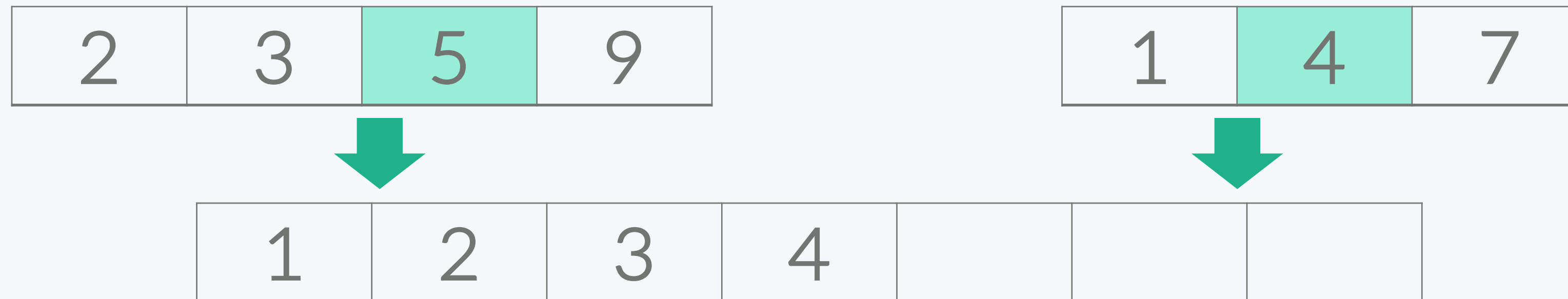


# 배열 합치기

28

<https://www.acmicpc.net/problem/11728>

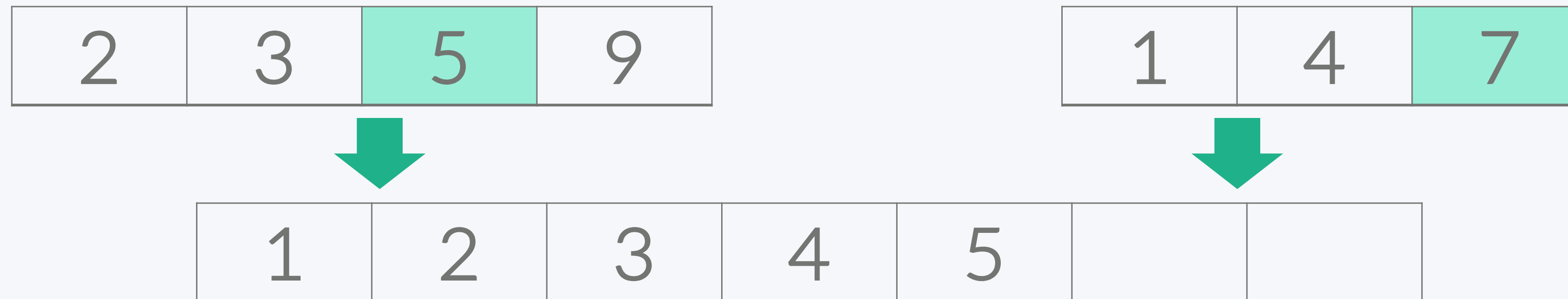
- 머지 소트에서 배열을 합치는 알고리즘 구현



# 배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



# 배열 합치기

<https://www.acmicpc.net/problem/11728>

30

- 머지 소트에서 배열을 합치는 알고리즘 구현

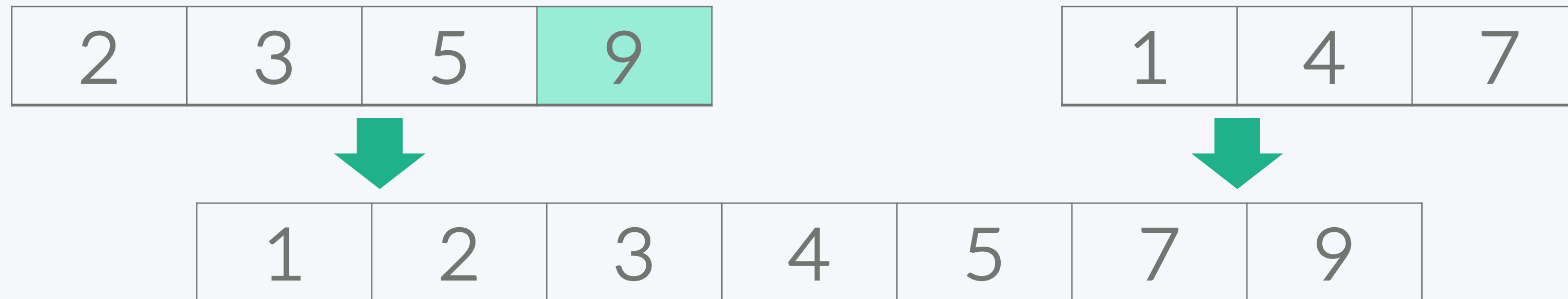


$O(N+M)$

# 배열 합치기

<https://www.acmicpc.net/problem/11728>

- 머지 소트에서 배열을 합치는 알고리즘 구현



# 배열 합치기

<https://www.acmicpc.net/problem/11728>

- C++: <https://gist.github.com/Baekjoon/e95f28c61154f6cdc3c4>
- C++: <https://gist.github.com/Baekjoon/bb3b4c3b2a896cd76dcc>
- Java: <https://gist.github.com/Baekjoon/c949446ce64f03a4f8727203066f2331>



# 문제 풀기

---

# 분할 정복

Divide & Conquer

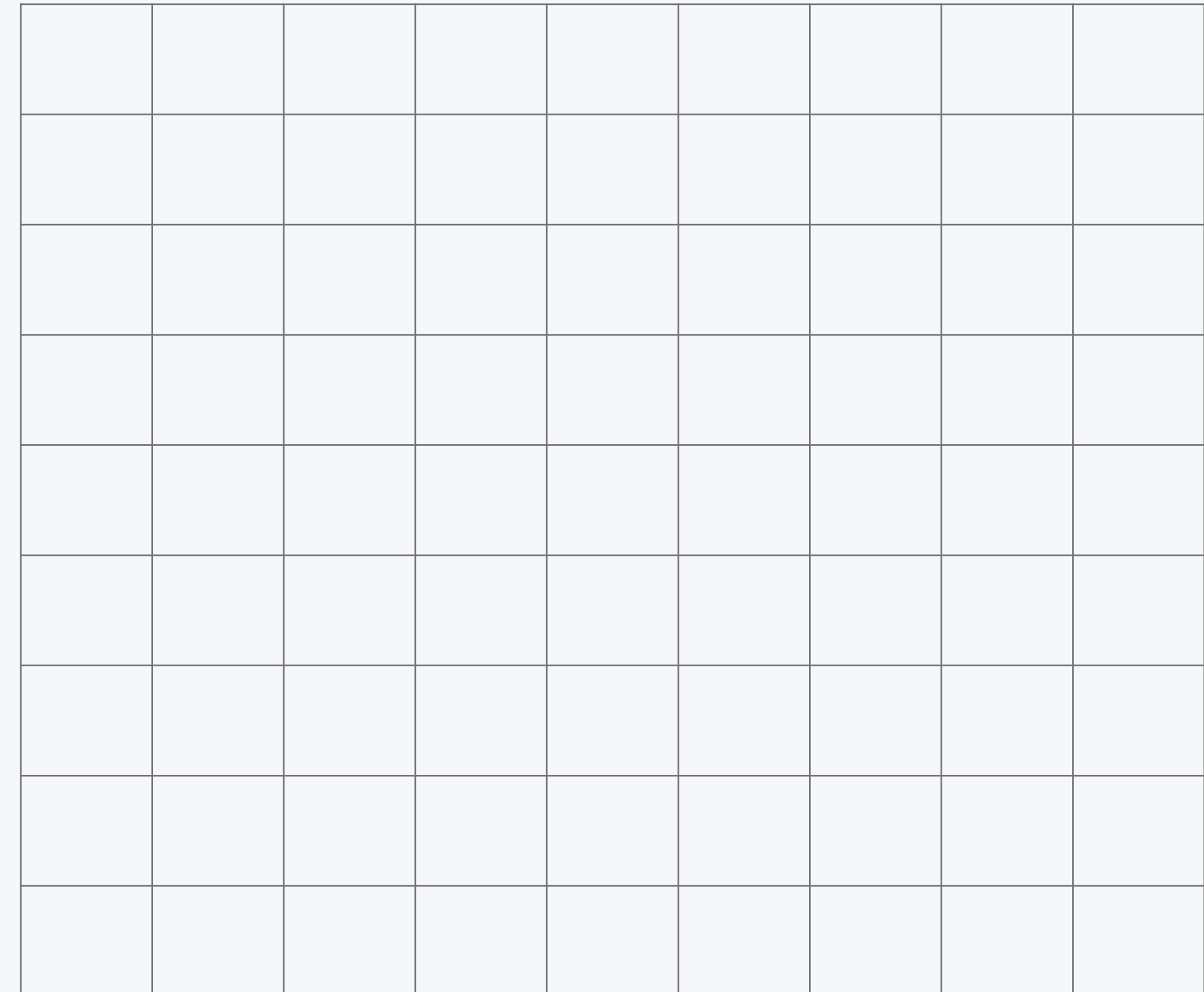
34

- 분할 정복 문제는 어떻게 함수를 만들어야 할지 결정해야 한다.
- 함수 -> 문제를 푸는 함수
- 그 함수 내에서는 작은 문제를 어떻게 호출해야 할지 결정
- 만약에 부분 문제의 정답을 합쳐야 하는 경우에는 합치는 것을 어떻게 해야 할지 결정

# 종이의 개수

<https://www.acmicpc.net/problem/1780>

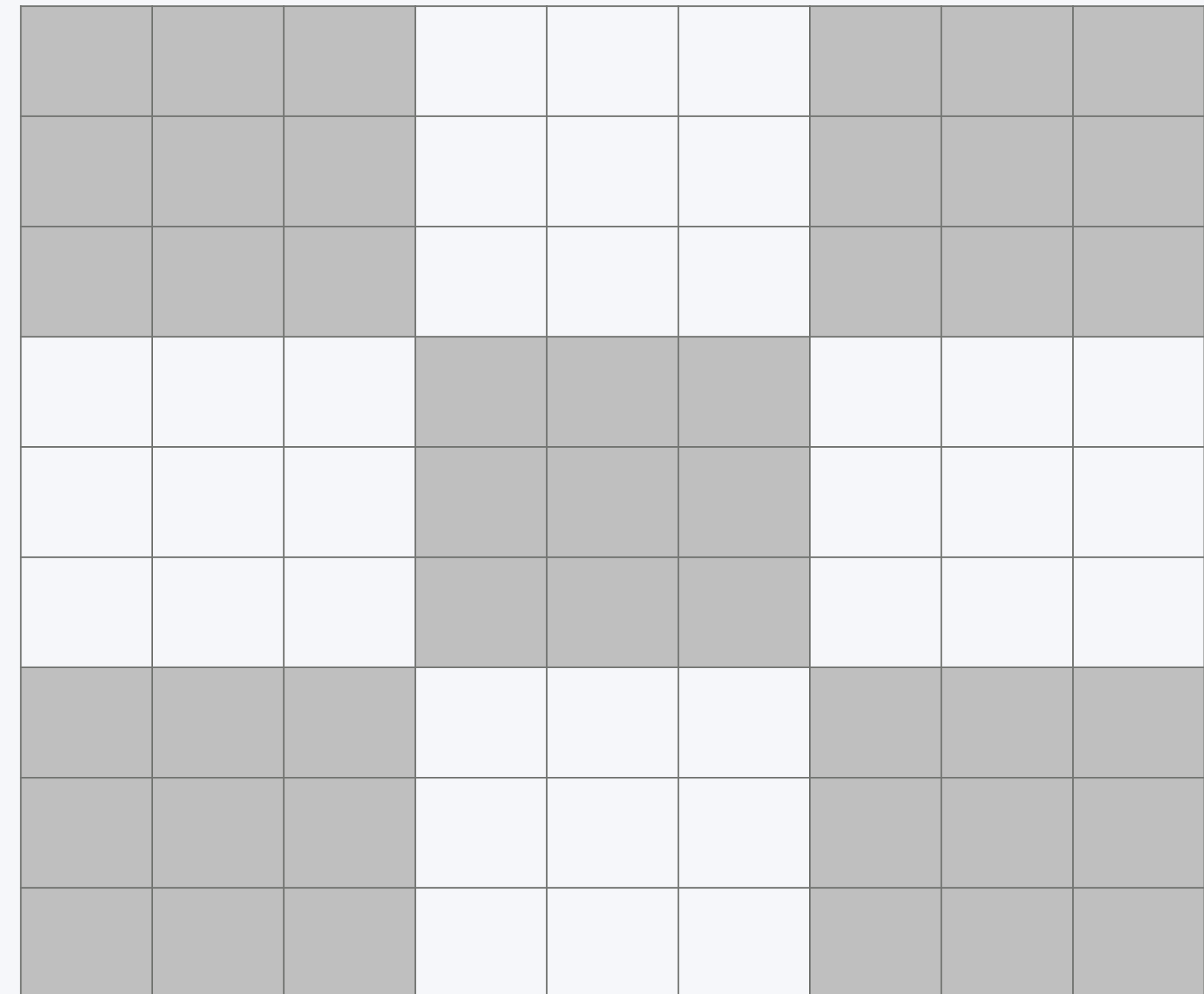
- `solve(0, 0, n)`
- $(0, 0)$ 부터 가로로  $n$ 개, 세로로  $n$ 개의 종이 개수를 확인하는 함수



# 종이의 개수

<https://www.acmicpc.net/problem/1780>

- $\text{solve}(x, y, n)$
- $(x, y)$ 부터 가로로  $n$ 개, 세로로  $n$ 개의 종이 개수를 확인하는 함수
- 작은 부분 문제는 총 9개
- $m = n/3$  이라고 했을 때
- 부분 문제를 나눠보면



# 종이의 개수

<https://www.acmicpc.net/problem/1780>

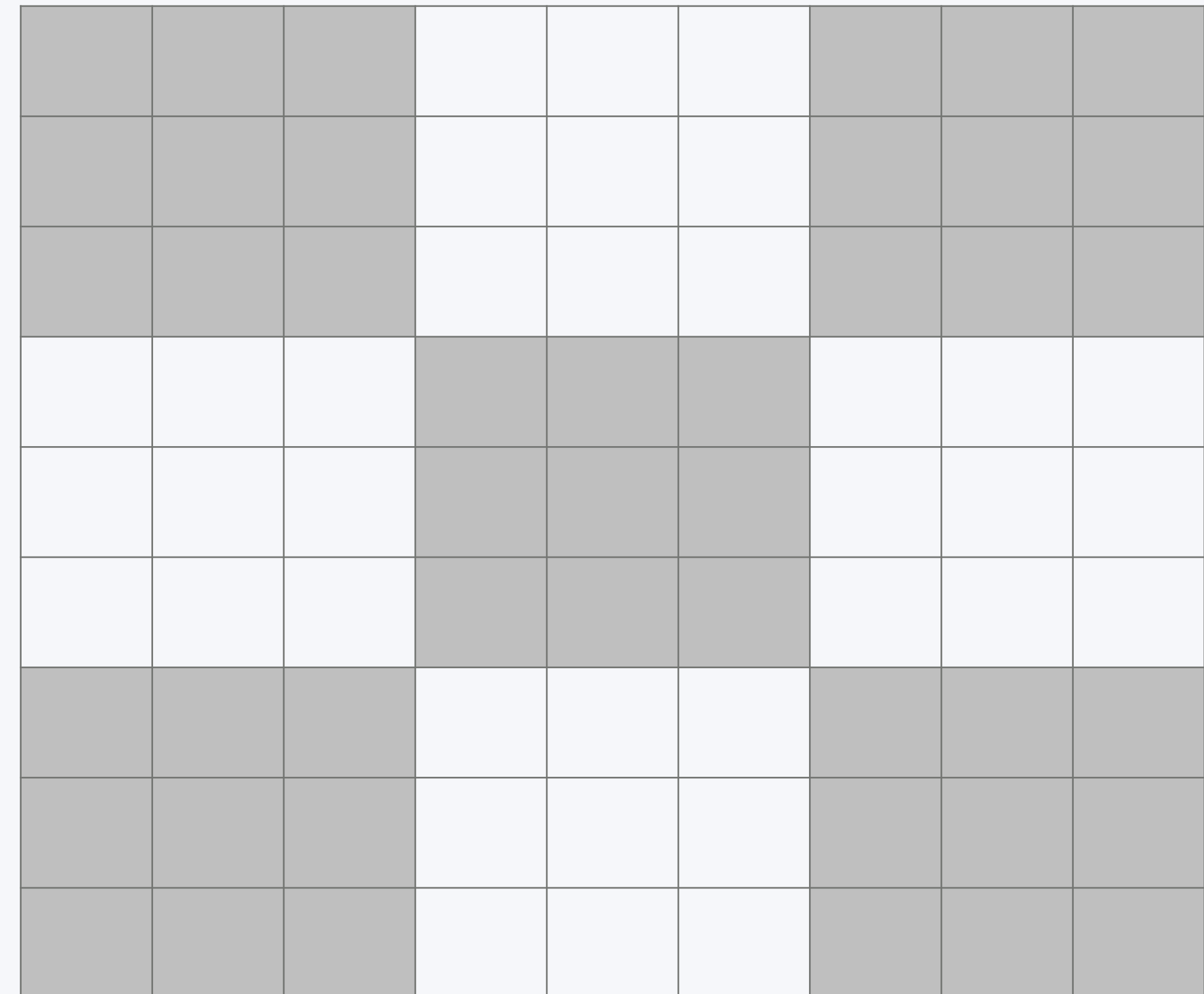
- $\text{solve}(x, y, n)$ 
  - 0:  $\text{solve}(x, y, m)$
  - 1:  $\text{solve}(x, y+m, m)$
  - 2:  $\text{solve}(x, y+2*m, m)$
  - 3:  $\text{solve}(x+m, y, m)$
  - 4:  $\text{solve}(x+m, y+m, m)$
  - 5:  $\text{solve}(x+m, y+2*m, m)$
  - 6:  $\text{solve}(x+2*m, y, m)$
  - 7:  $\text{solve}(x+2*m, y+m, m)$
  - 8:  $\text{solve}(x+2*m, y+2*m, m)$

0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
0	0	0	1	1	1	2	2	2
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
3	3	3	4	4	4	5	5	5
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8
6	6	6	7	7	7	8	8	8

# 종이의 개수

<https://www.acmicpc.net/problem/1780>

- $\text{solve}(x, y, n)$
- $(x, y)$ 부터 가로로  $n$ 개, 세로로  $n$ 개의 종이 개수를 확인하는 함수
- 부분 문제를 호출하기 전에
- 같은 수로 되어 있는지를 확인해야 한다
- 그게 아니면 부분 문제를 호출



# 종이의 개수

<https://www.acmicpc.net/problem/1780>

```
void solve(int x, int y, int n) {
    if (same(x, y, n)) {
        cnt[a[x][y]+1] += 1;
        return;
    }
    int m = n/3;
    for (int i=0; i<3; i++) {
        for (int j=0; j<3; j++) {
            solve(x+i*m, y+j*m, m);
        }
    }
}
```

# 종이의 개수

<https://www.acmicpc.net/problem/1780>

```
bool same(int x, int y, int n) {  
    for (int i=x; i<x+n; i++) {  
        for (int j=y; j<y+n; j++) {  
            if (a[x][y] != a[i][j]) {  
                return false;  
            }  
        }  
    }  
    return true;  
}
```



# 종이의 개수

<https://www.acmicpc.net/problem/1780>

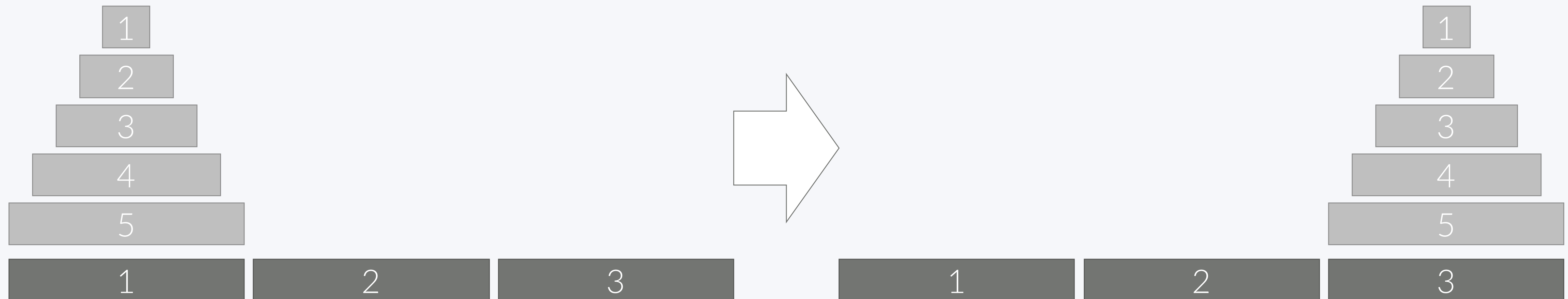
- C/C++: <https://gist.github.com/Baekjoon/06950b687d63facf0e01>
- Java: <https://gist.github.com/Baekjoon/d699b334c05b743006867a4e54272c76>

# 하노이 탑 이동 순서

42

<https://www.acmicpc.net/problem/11729>

- 하노이 탑 문제
- 규칙 1: 한 번에 한 개의 원판만 다른 탑으로 옮길 수 있다
- 규칙 2: 쌓아 놓은 원판은 항상 위의 것이 아래의 것보다 작아야 한다 (중간 과정 포함)



# 하노이 탑 이동 순서

<https://www.acmicpc.net/problem/11729>

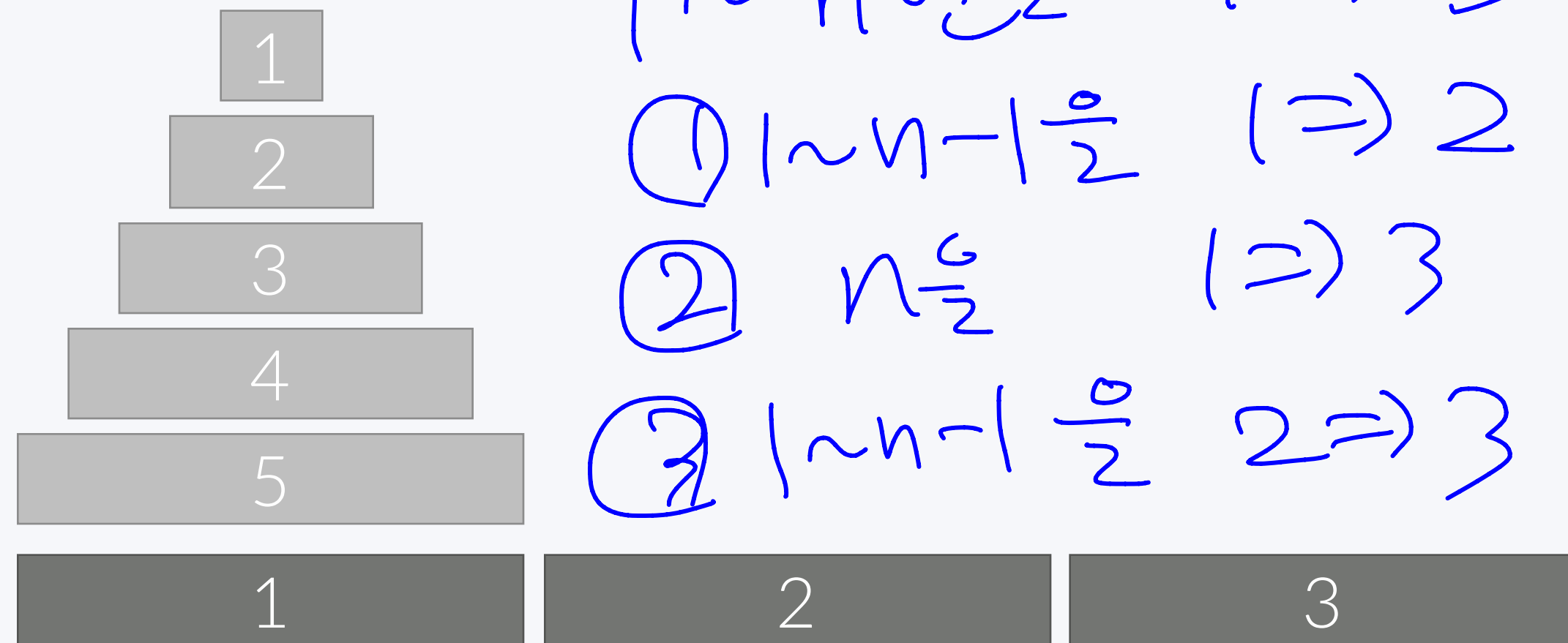
1 ~ 5 번 원판을 1에서 3으로

① 1 ~ 4 번 원판을 1에서 2로

② 5 번을 1에서 3으로

③ 1 ~ 4 번 원판을 2에서 3으로

43

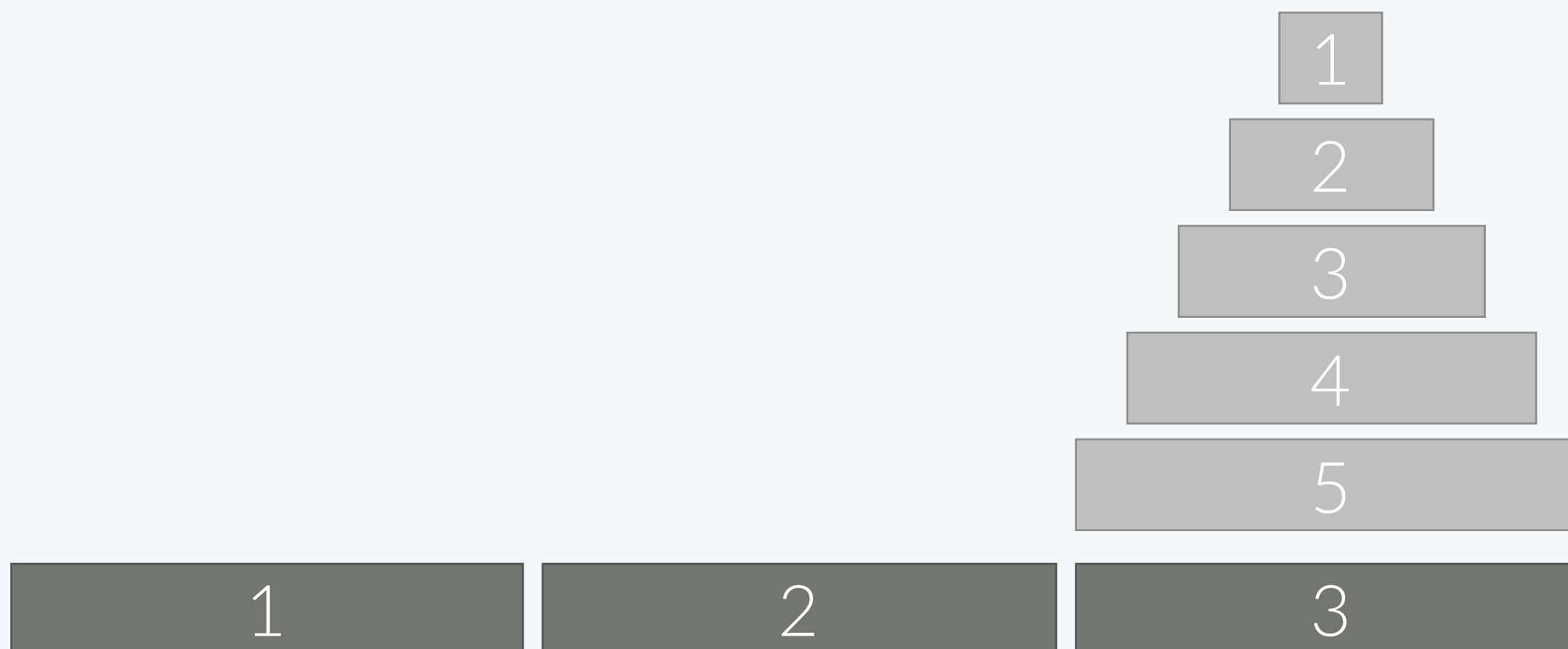
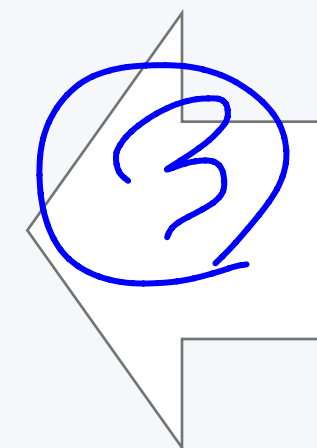
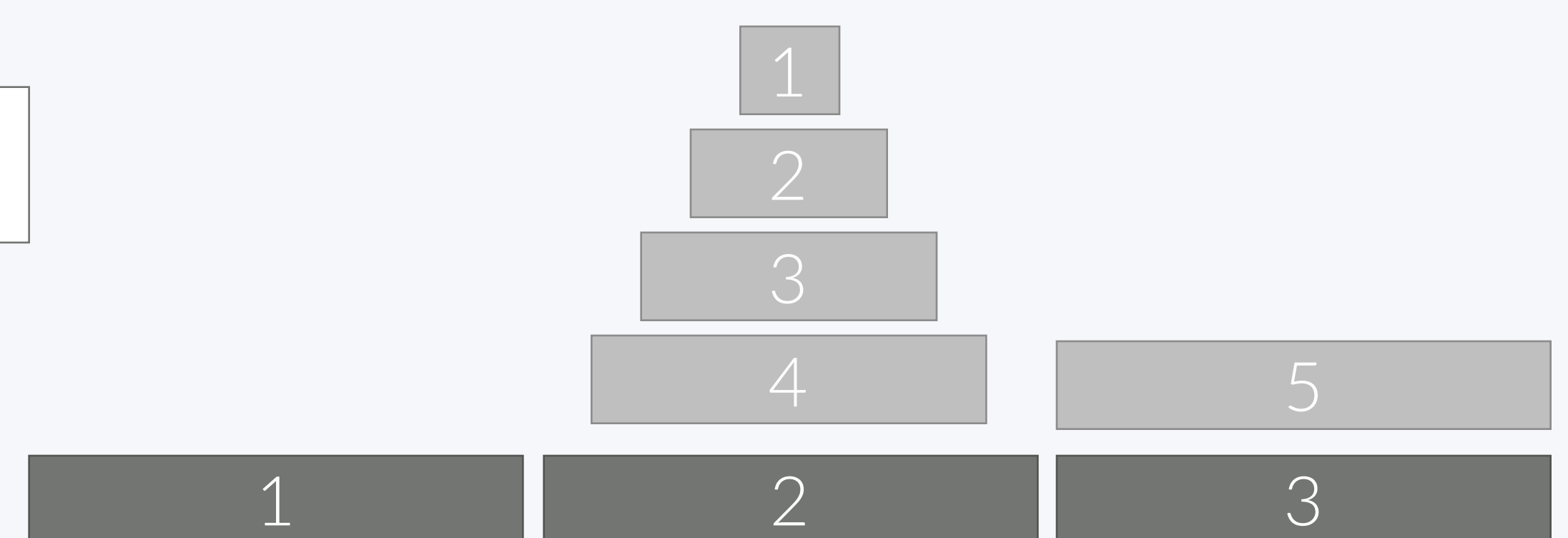
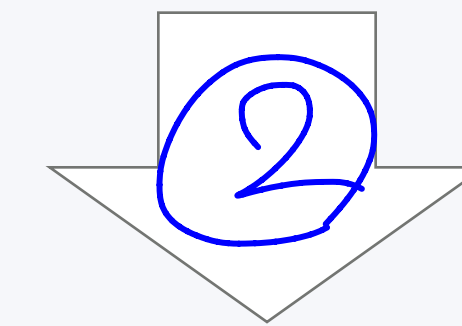
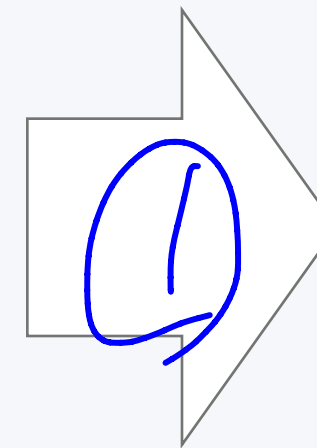


1 ~ n 번을 1  $\Rightarrow$  3

① 1 ~ n-1 을 1  $\Rightarrow$  2

② n 을 1  $\Rightarrow$  3

③ 1 ~ n-1 을 2  $\Rightarrow$  3



# 하노이 탑 이동 순서

<https://www.acmicpc.net/problem/11729>

- $\text{solve}(n, x, y)$
- 1~n개의 원판을 x에서 y로 이동하는 함수
- 1~n-1개의 원판을 x에서 z로 이동한다. (z는 x와 y가 아닌 원판)
- n번 원판을 x에서 y로 이동한다.
- 1~n-1개의 원판을 z에서 y로 이동한다.

$\text{Solve}(n, x, y)$   
1~n번 원판을 x에서 y로

①  $\text{Solve}(n-1, x, z)$

② n을  $x \Rightarrow y$

③  $\text{Solve}(n-1, z, y)$

# 하노이 탑 이동 순서

<https://www.acmicpc.net/problem/11729>

- $\text{solve}(n, x, y)$
- 1~n개의 원판을 x에서 y로 이동하는 함수
- 1~n-1개의 원판을 x에서 z로 이동한다. (z는 x와 y가 아닌 원판)
  - $\text{solve}(n-1, x, z)$
- n번 원판을 x에서 y로 이동한다.
- 1~n-1개의 원판을 z에서 y로 이동한다.
  - $\text{solve}(n-1, z, y)$

# 하노이 탑 이동 순서

46

<https://www.acmicpc.net/problem/11729>

```
void solve(int n, int x, int y) {  
    if (n == 0) return;  
    solve(n-1, x, 6-x-y);  
    printf("%d %d\n", x, y);  
    solve(n-1, 6-x-y, y);  
}
```

D[i] = 1>1 하노이 탑 0/2 3/1

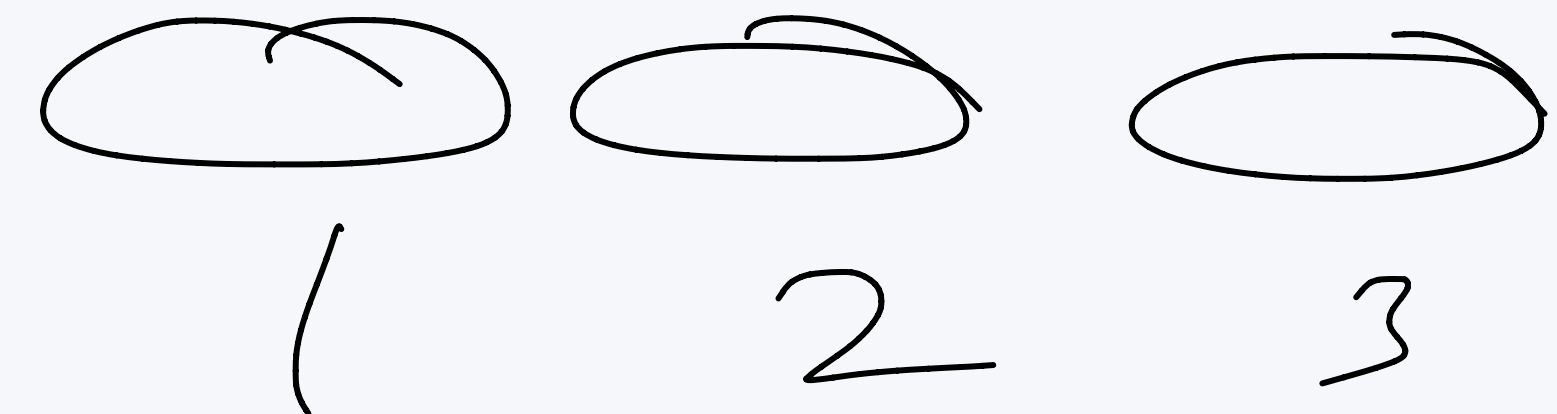
D[1] = (

$$D[i] = D[i-1] \times 2 + 1 = 2^i - 1$$

$$z = 6 - x - y$$

↪ 중간지점

1 3 17 15 31 63 127



# 하노이 탑 이동 순서

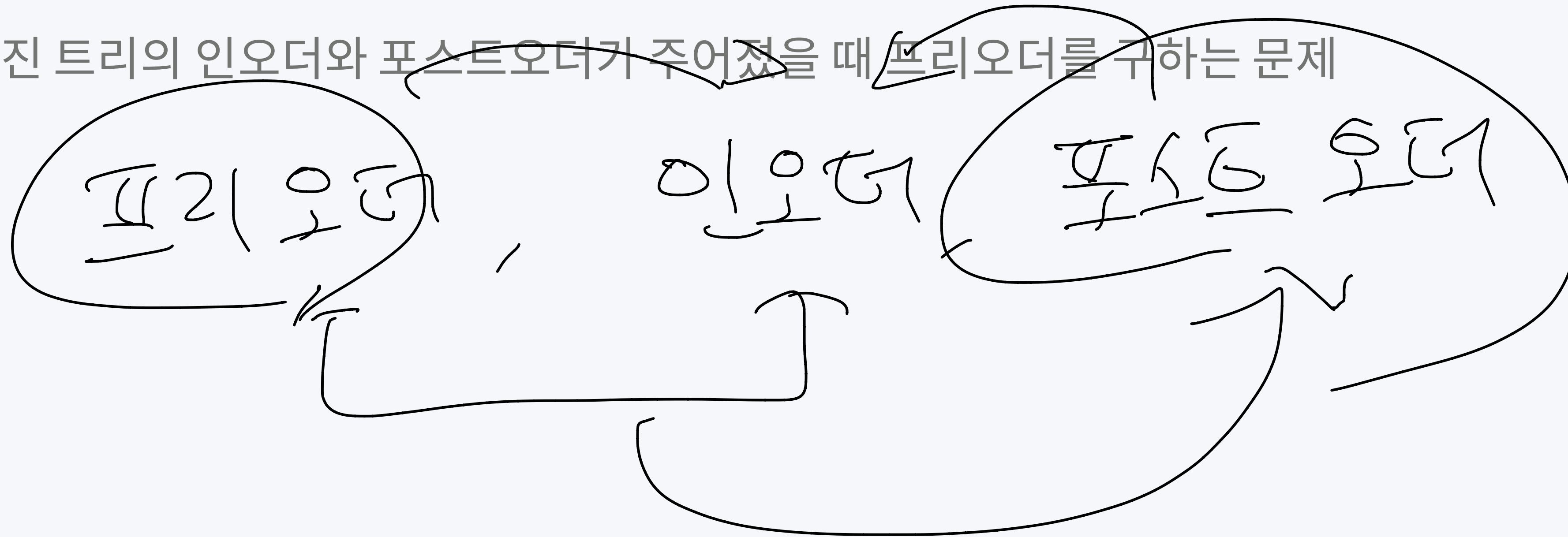
<https://www.acmicpc.net/problem/11729>

- C/C++: <https://gist.github.com/Baekjoon/38c6cf2fe47d3f4055cf>
- Java: <https://gist.github.com/Baekjoon/2101ae21ba1429047c394b2026d723ba>

# 트리의 순회

<https://www.acmicpc.net/problem/2263>

- N개의 정점을 갖는 이진 트리의 정점에 1부터 N까지 번호가 중복없이 매겨져 있다
- 이 이진 트리의 인오더와 포스트오더가 주어졌을 때 프리오더를 구하는 문제





# 트리의 순회

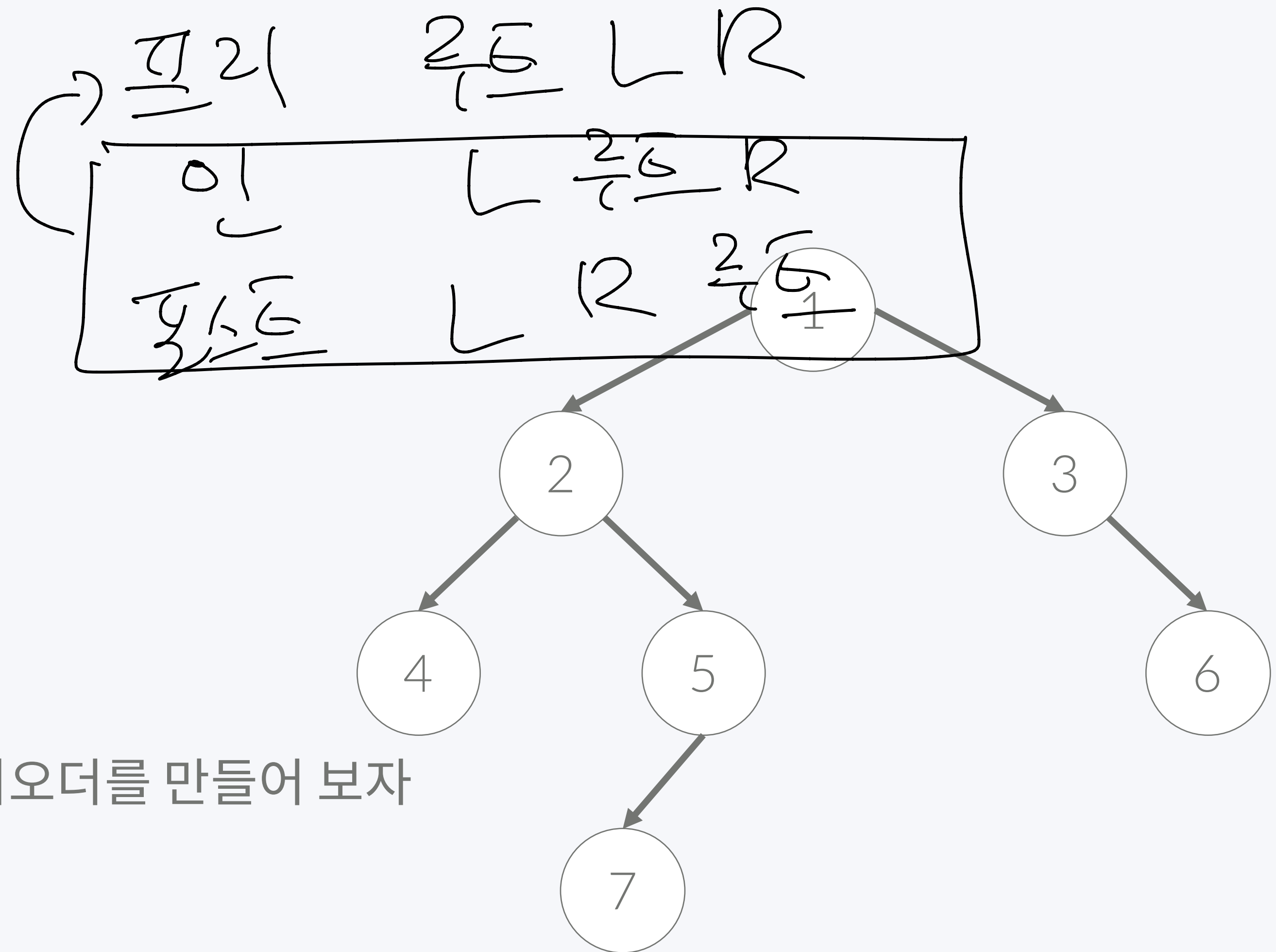
<https://www.acmicpc.net/problem/2263>

• 프리오더: 1 2 4 5 7 3 6 ✓

• 인오더: 4 2 7 5 1 3 6

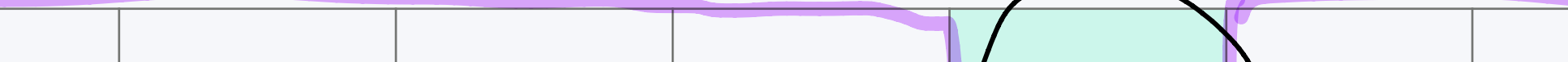
• 포스트오더: 4 7 5 2 6 3 1

• 인오더와 포스트오더를 가지고 프리오더를 만들어 보자



$$O_L: L \xrightarrow{\frac{2}{1} \underline{\underline{E}}} R$$

- L R



4	2	7	5	1	3	6
---	---	---	---	---	---	---

4	7	5	2	6	3	1
---	---	---	---	---	---	---

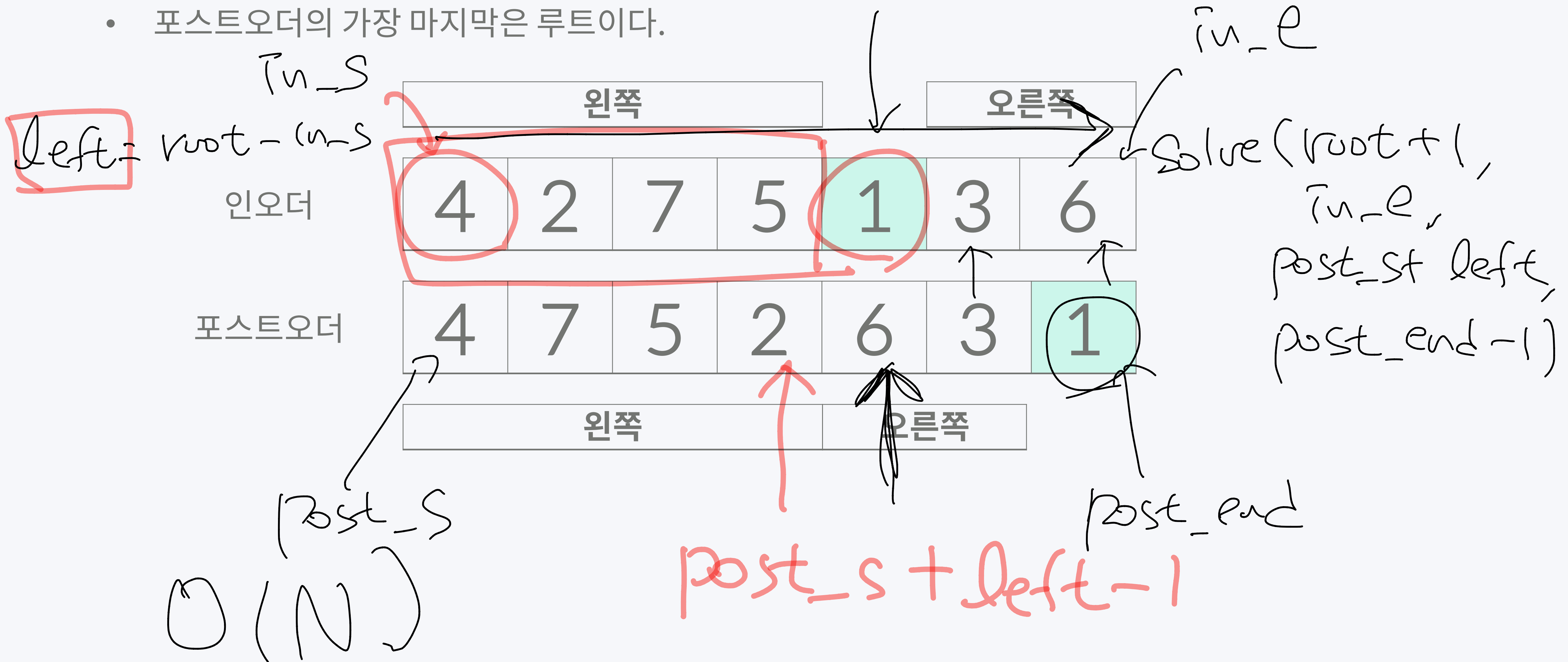
II: L R  $\frac{2}{1}$  E

# 트리의 순회

<https://www.acmicpc.net/problem/2263>

Solve(in\_s, root-1, post\_s, post\_s  
+ len-1)  
root

- 포스트오더의 가장 마지막은 루트이다.

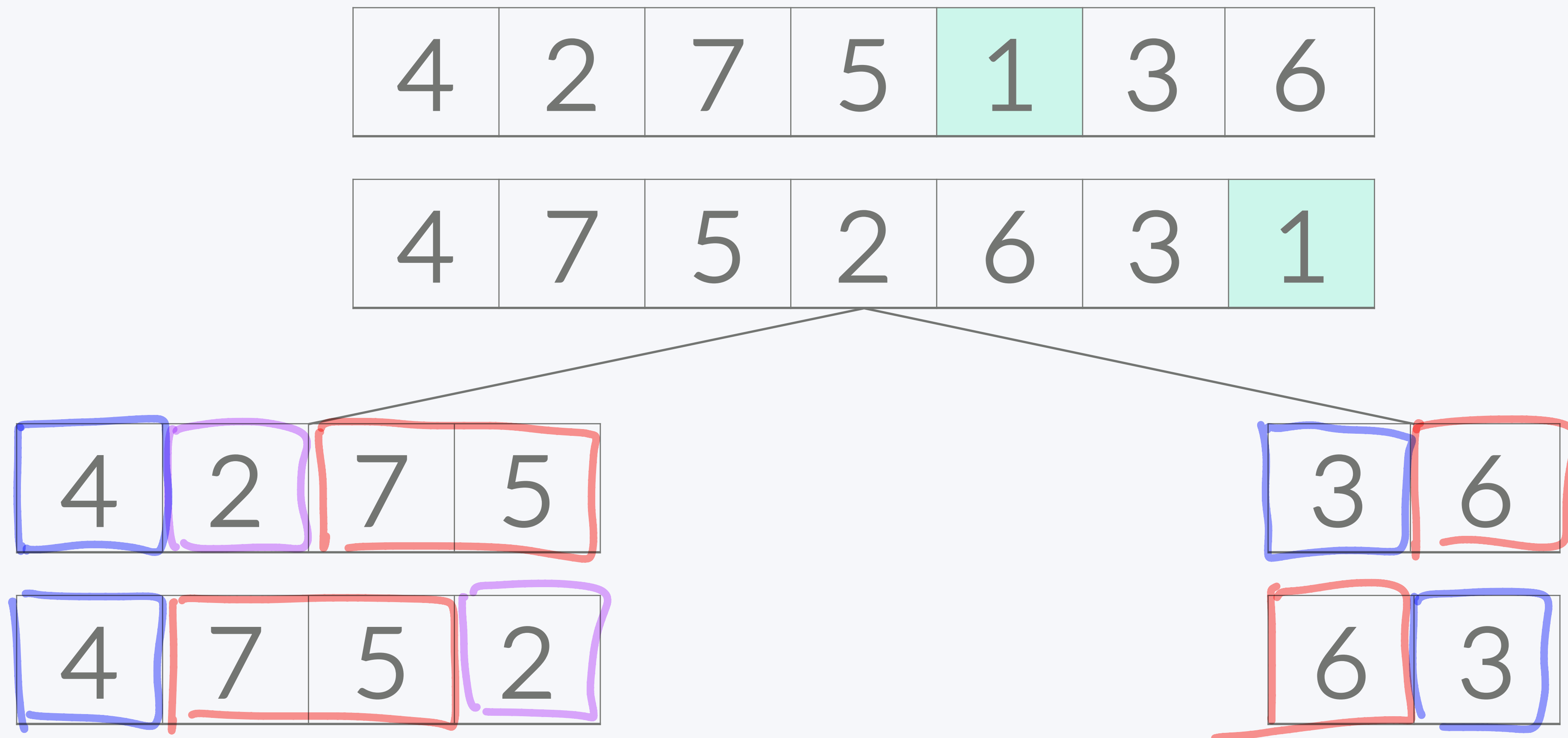


o(n)

# 트리의 순회

<https://www.acmicpc.net/problem/2263>

- 왼쪽과 오른쪽으로 나뉘서 풀 수 있다.



# 트리의 순회

53

<https://www.acmicpc.net/problem/2263>

- C/C++: <https://gist.github.com/Baekjoon/8244aa7f00e2352cc78f462c4b9bbecc>

# 쿼드트리

<https://www.acmicpc.net/problem/1992>

- 쿼드트리로 압축하여 출력하는 문제
- 분할 정복 + 프리오더
- 색종이 나누기와 비슷한 과정을 거치면 된다

# 쿼드트리

<https://www.acmicpc.net/problem/1992>

```
void solve(int x, int y, int n) {
    if (same(x,y,n)) {
        printf("%d",a[x][y]);
    } else {
        printf("(");
        int m = n/2;
        for (int i=0; i<2; i++) {
            for (int j=0; j<2; j++) {
                solve(x+m*i, y+m*j, m);
            }
        }
        printf(")");
    }
}
```

# 쿼드트리

<https://www.acmicpc.net/problem/1992>

- C/C++: <https://gist.github.com/Baekjoon/0e2f73e53059d2b345ff>



# 별찍기 - 10

57

<https://www.acmicpc.net/problem/2447>

- 분할 정복으로 멋지게 별을 찍는 문제

# 별찍기 - 10

<https://www.acmicpc.net/problem/2447>

- 가운데는 빈칸이고, 나머지 3칸은 다시 재귀적인 구조를 가지게 된다.

	재			귀			재	
			빈	칸	빈			
	귀		칸	빈	칸		귀	
			빈	칸	빈			
	재			귀			재	

# 별찍기 - 10

59

<https://www.acmicpc.net/problem/2447>

- C/C++: <https://gist.github.com/Baekjoon/7267452f6a927560e57b>

# 별찍기 - 11

60

<https://www.acmicpc.net/problem/2448>

- 분할 정복으로 멋지게 별을 찍는 문제

# 별찍기 - 11

<https://www.acmicpc.net/problem/2448>

- 별찍기 - 10과 비슷하지만, 삼각형으로 계산하는 문제

# 별찍기 - 11

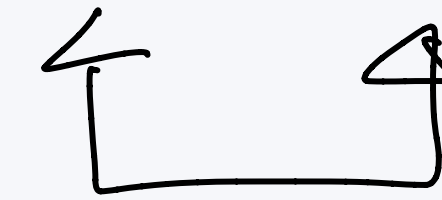
62

<https://www.acmicpc.net/problem/2448>

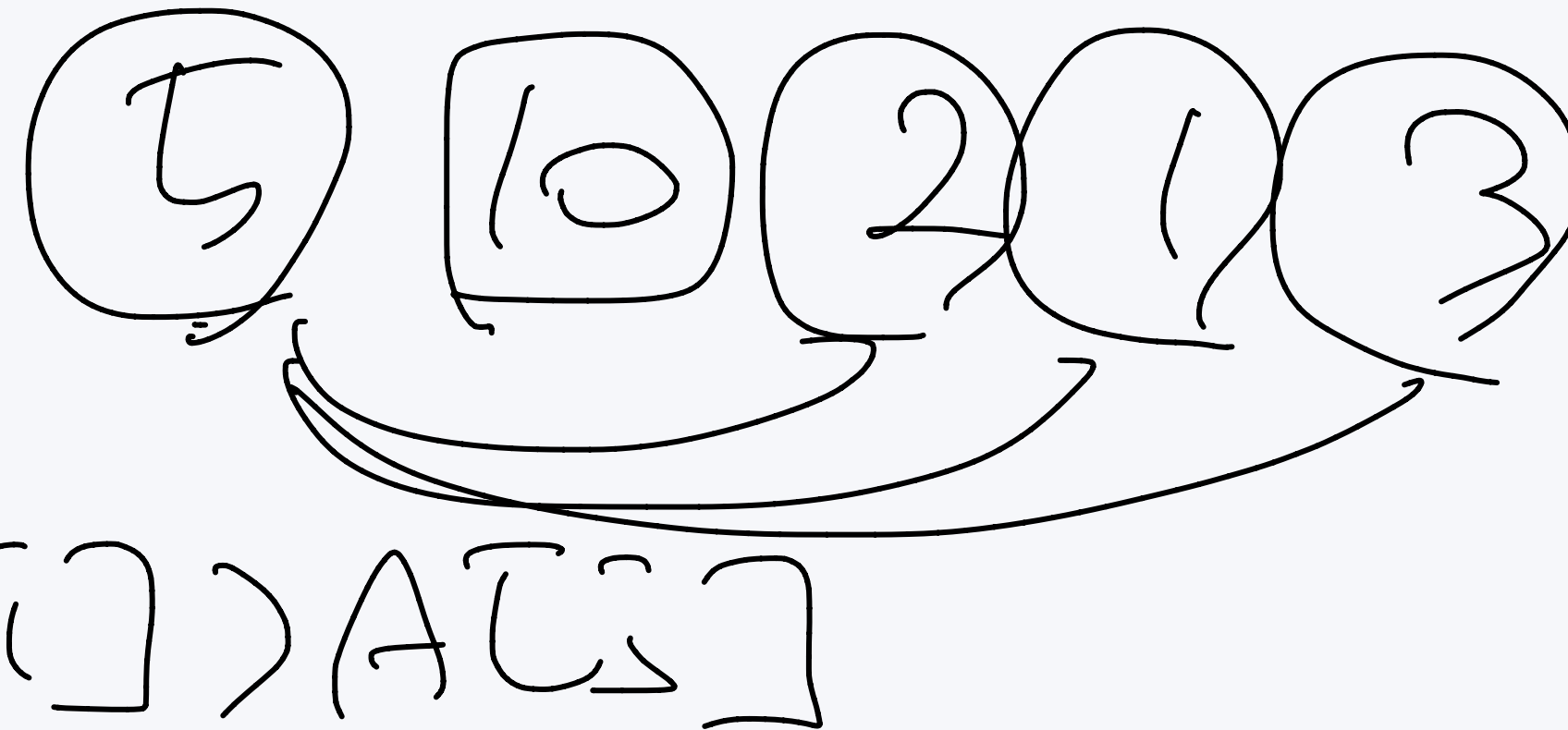
- C/C++: <https://gist.github.com/Baekjoon/be9211f28bbd492fa09a>

# 버블 소트

<https://www.acmicpc.net/problem/1517>



- N개로 이루어진 수열  $A[1], A[2], \dots, A[N]$ 이 있을 때
- Swap이 총 몇 번 발생하는지 알아내는 문제
- $3\ 2\ 1 \rightarrow 2\ 3\ 1 \rightarrow 2\ 1\ 3 \rightarrow 1\ 2\ 3$  (3번)

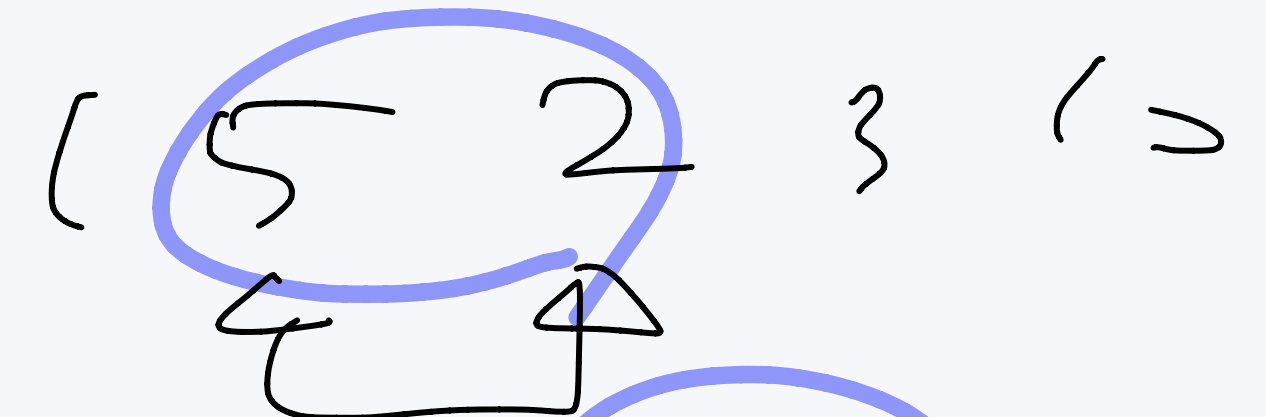
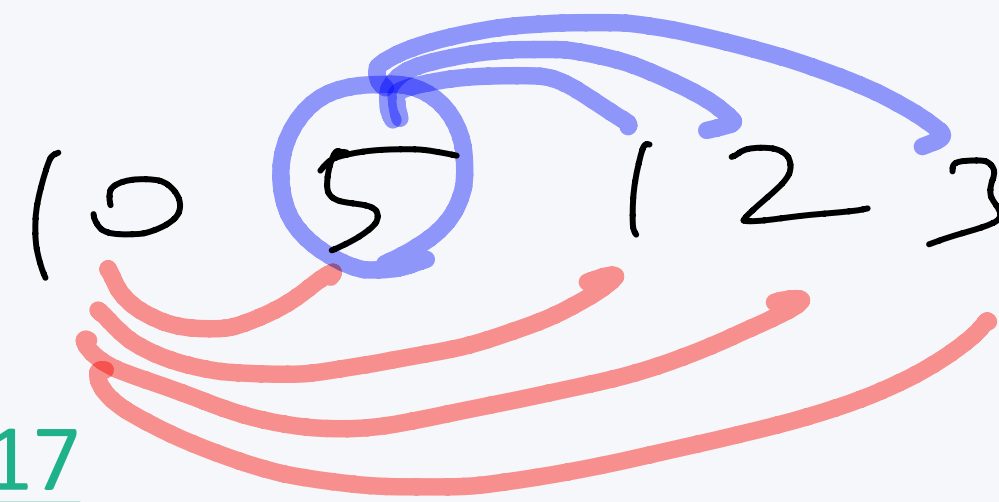
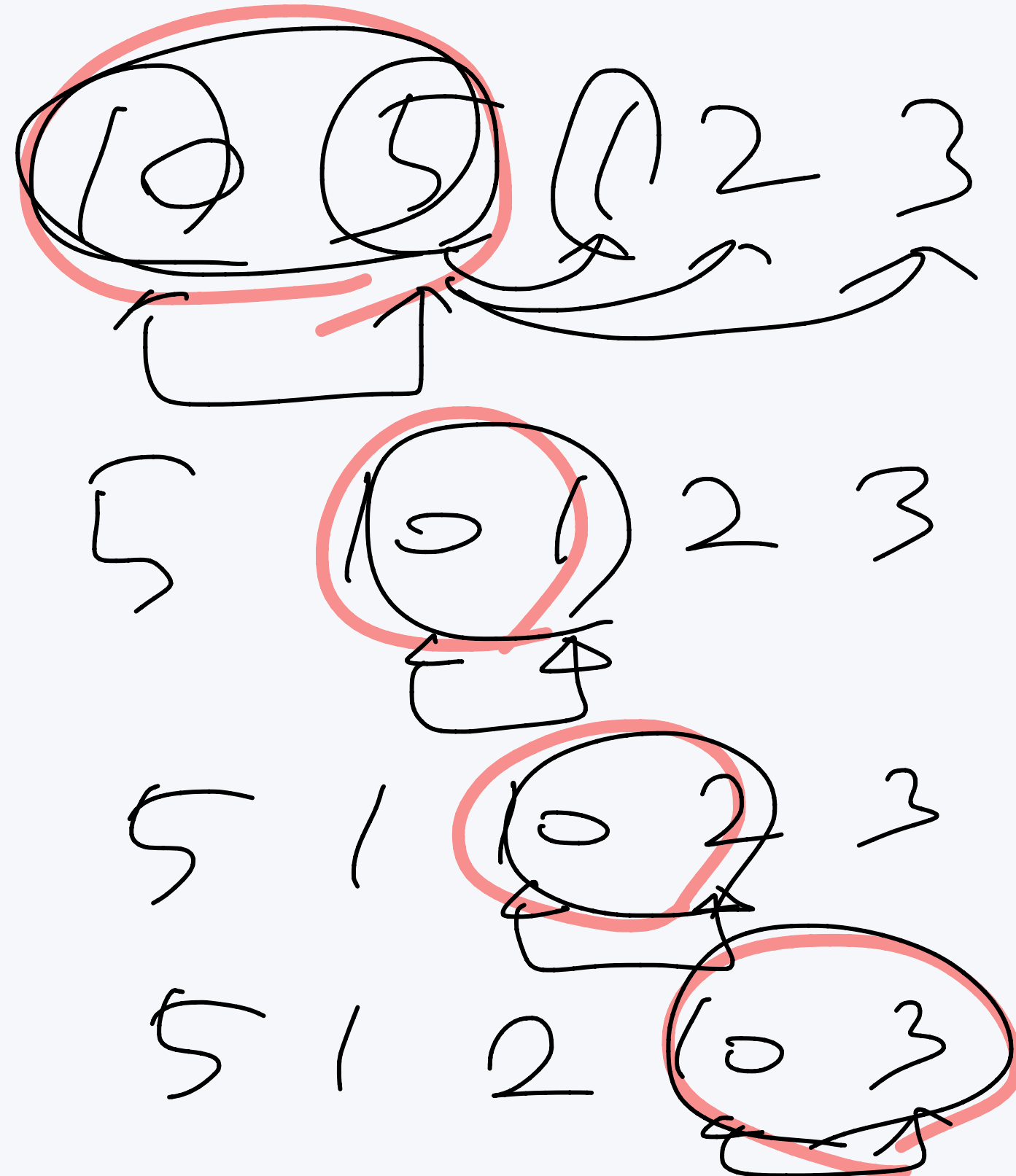


$i < j$ ,  $A[i] > A[j]$

# 버블 소트

<https://www.acmicpc.net/problem/1517>

- 이 문제는 수열에서 inversion의 개수를 세는 문제이다.
- inversion:  $A[i] > A[j]$  ( $i < j$ )
- 머지 소트를 하면서 문제를 풀 수 있다.



$N \leq 5000$

5 1 2 3 10

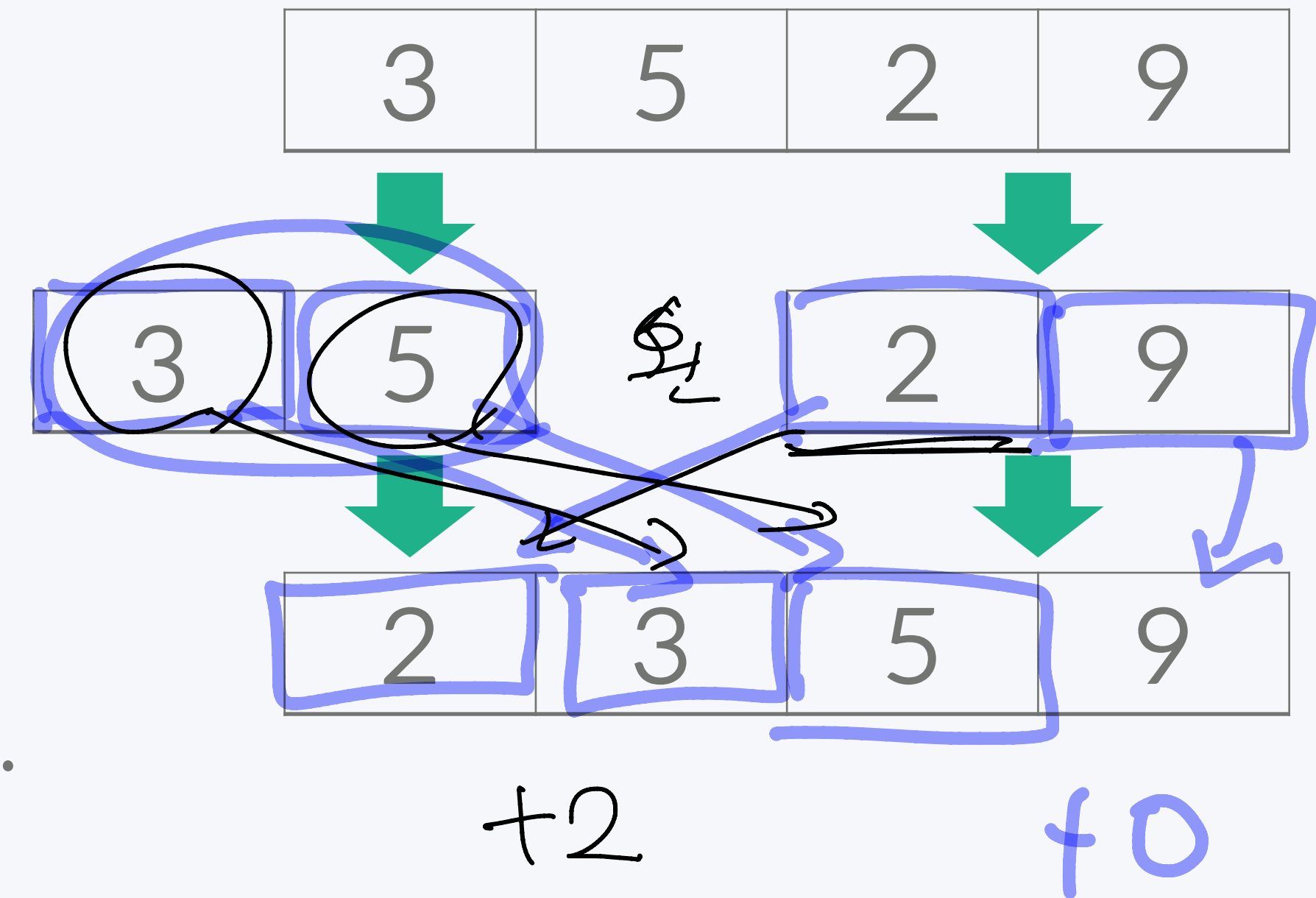


# 버블 소트

65

<https://www.acmicpc.net/problem/1517>

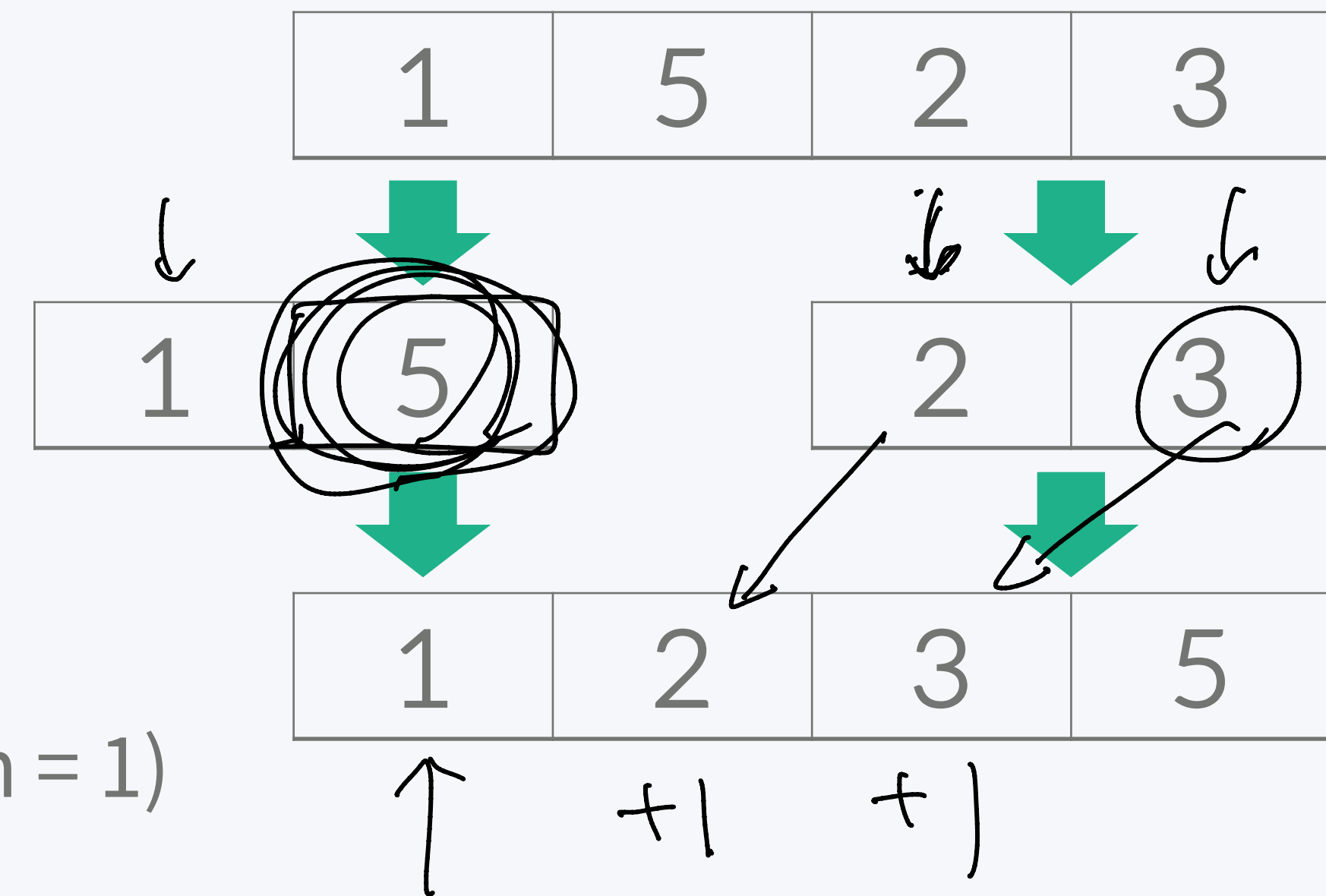
- 3 5 2 9의 inversion의 개수는
- $3 > 2, 5 > 2$ 로 2개다.
- 오른쪽 절반이 이동할 때
- 왼쪽 절반에서 아직 이동하지 않은 것의 개수가
- 그 때의 inversion의 개수이다.
- 2가 먼저 이동하는데, 그 때 왼쪽 절반에는 3과 5가 있다.
- 이것은  $3 > 2, 5 > 2$ 를 의미한다.



# 버블 소트

<https://www.acmicpc.net/problem/1517>

- 1 5 2 3의 inversion의 개수는
- $5 > 2, 5 > 3$ 로 2개다.
- 오른쪽 절반이 이동할 때
- 왼쪽 절반에서 아직 이동하지 않은 것의 개수가
- 그 때의 inversion의 개수이다.
- 2가 이동할 때, 왼쪽 절반에는 5가 남아있다. (inversion = 1)
- 3이 이동할 때도 왼쪽 절반에는 5가 남아있다. (inversion = 1)



# 버블 소트

<https://www.acmicpc.net/problem/1517>

```
long long solve(int start, int end) {  
    if (start == end) {  
        return 0;  
    }  
    int mid = (start+end)/2;  
    long long ans = solve(start, mid) + solve(mid+1, end);  
    int i = start;  
    int j = mid+1;  
    int k = 0;
```

# 버블 소트

<https://www.acmicpc.net/problem/1517>

```
while (i <= mid || j <= end) {
    if (i <= mid && (j > end || a[i] <= a[j])) {
        b[k++] = a[i++];
    } else {
        ans += (long long)(mid-i+1);
        b[k++] = a[j++];
    }
}

for (int i=start; i<=end; i++) {
    a[i] = b[i-start];
}

return ans;
}
```

# 버블 소트

<https://www.acmicpc.net/problem/1517>

- C/C++: <https://gist.github.com/Baekjoon/5e99ccde4b05b760d7d8>
- Java: <https://gist.github.com/Baekjoon/bfe783fdd0e7a8973735762e05e4d51c>

# 가장 가까운 두 점

70

<https://www.acmicpc.net/problem/2261>

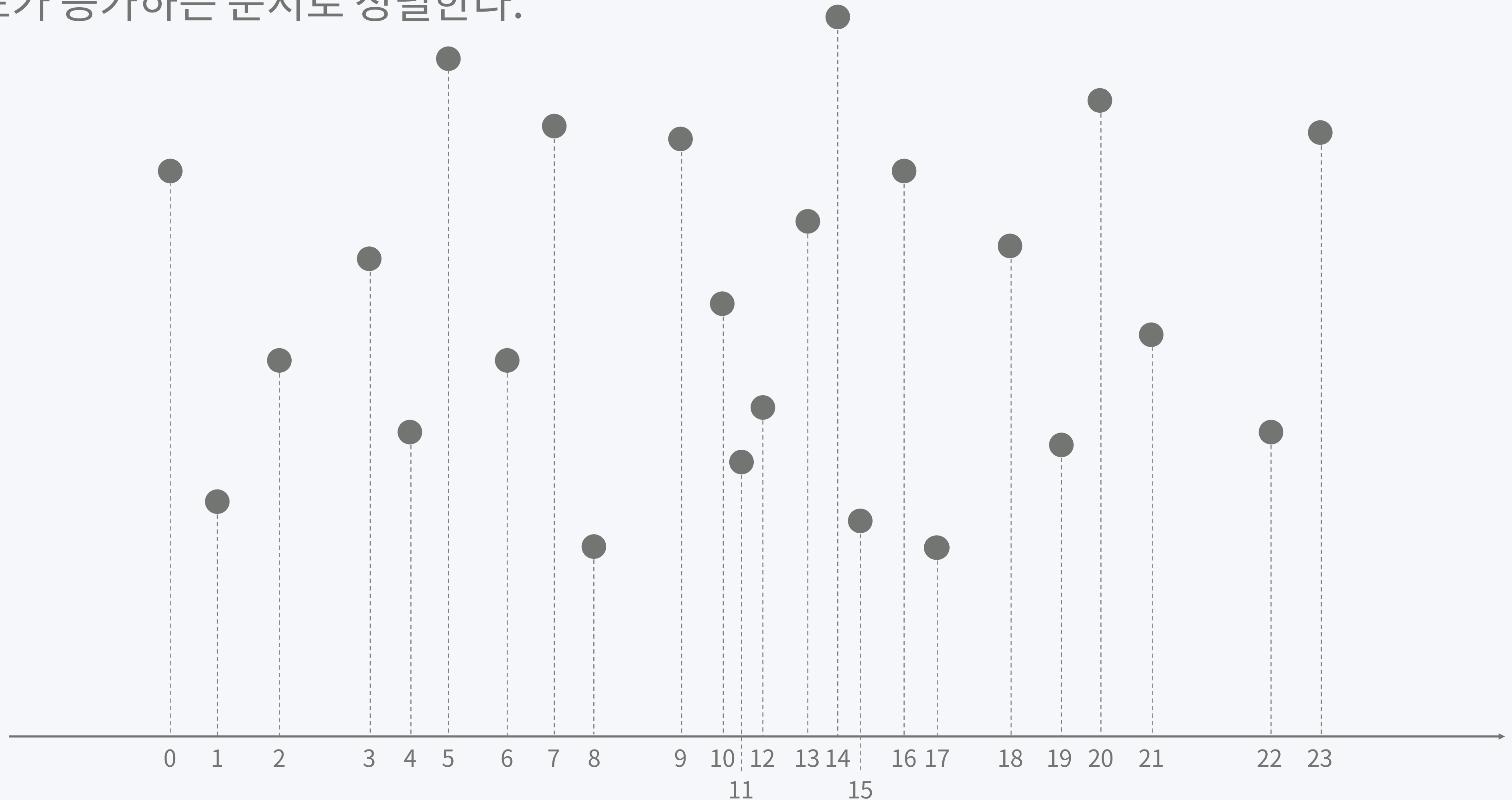
- 2차원 평면 위의 N개의 점 중에서 가장 가까운 두 점을 찾는 문제
- $2 \leq N \leq 100,000$

# 가장 가까운 두 점

71

<https://www.acmicpc.net/problem/2261>

- 먼저 점을 x좌표가 증가하는 순서로 정렬한다.

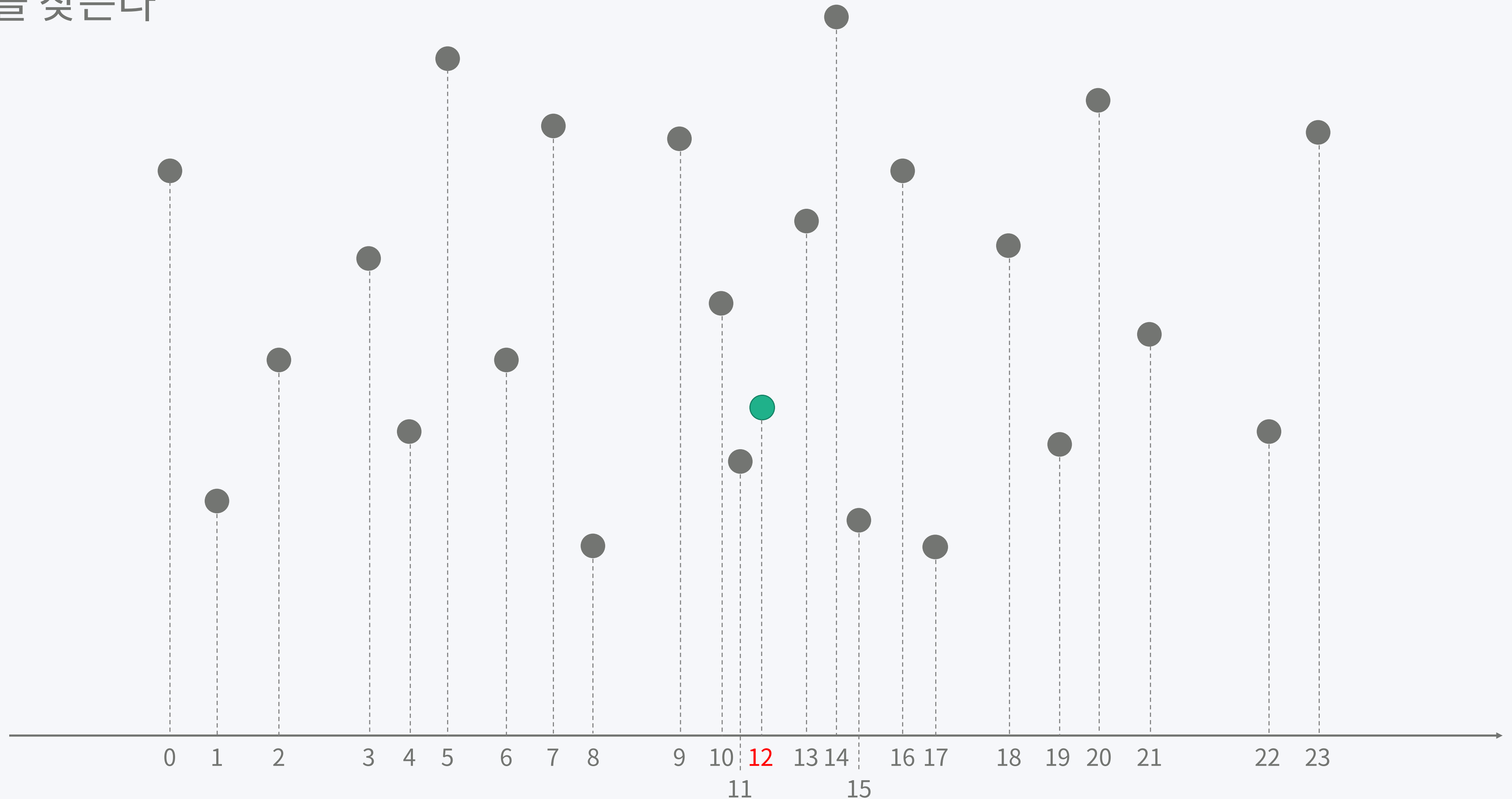


# 가장 가까운 두 점

72

<https://www.acmicpc.net/problem/2261>

- 중간에 있는 점을 찾는다



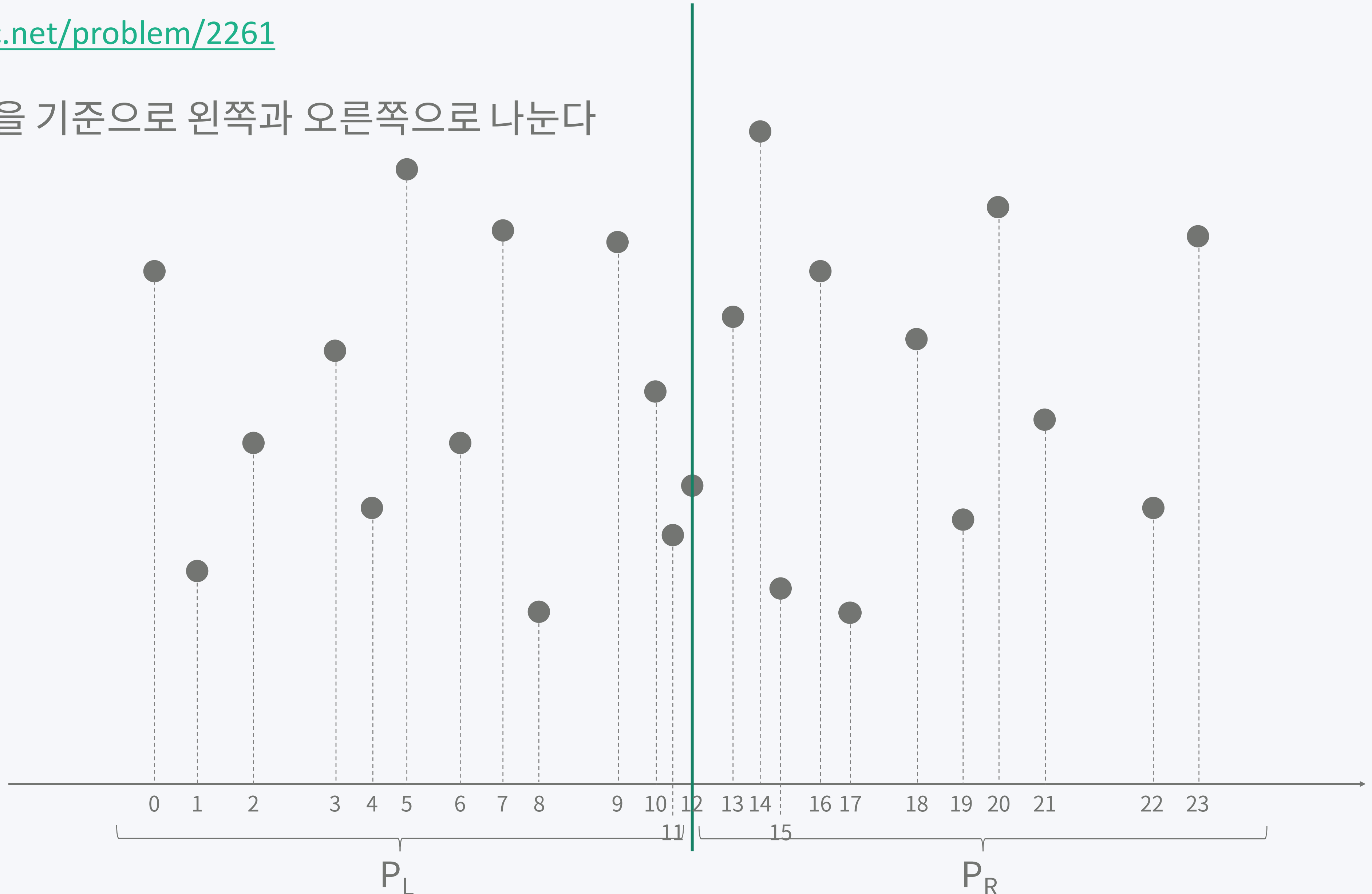


# 가장 가까운 두 점

73

<https://www.acmicpc.net/problem/2261>

- 중간에 있는 점을 기준으로 왼쪽과 오른쪽으로 나눈다
- 왼쪽:  $P_L$
- 오른쪽:  $P_R$



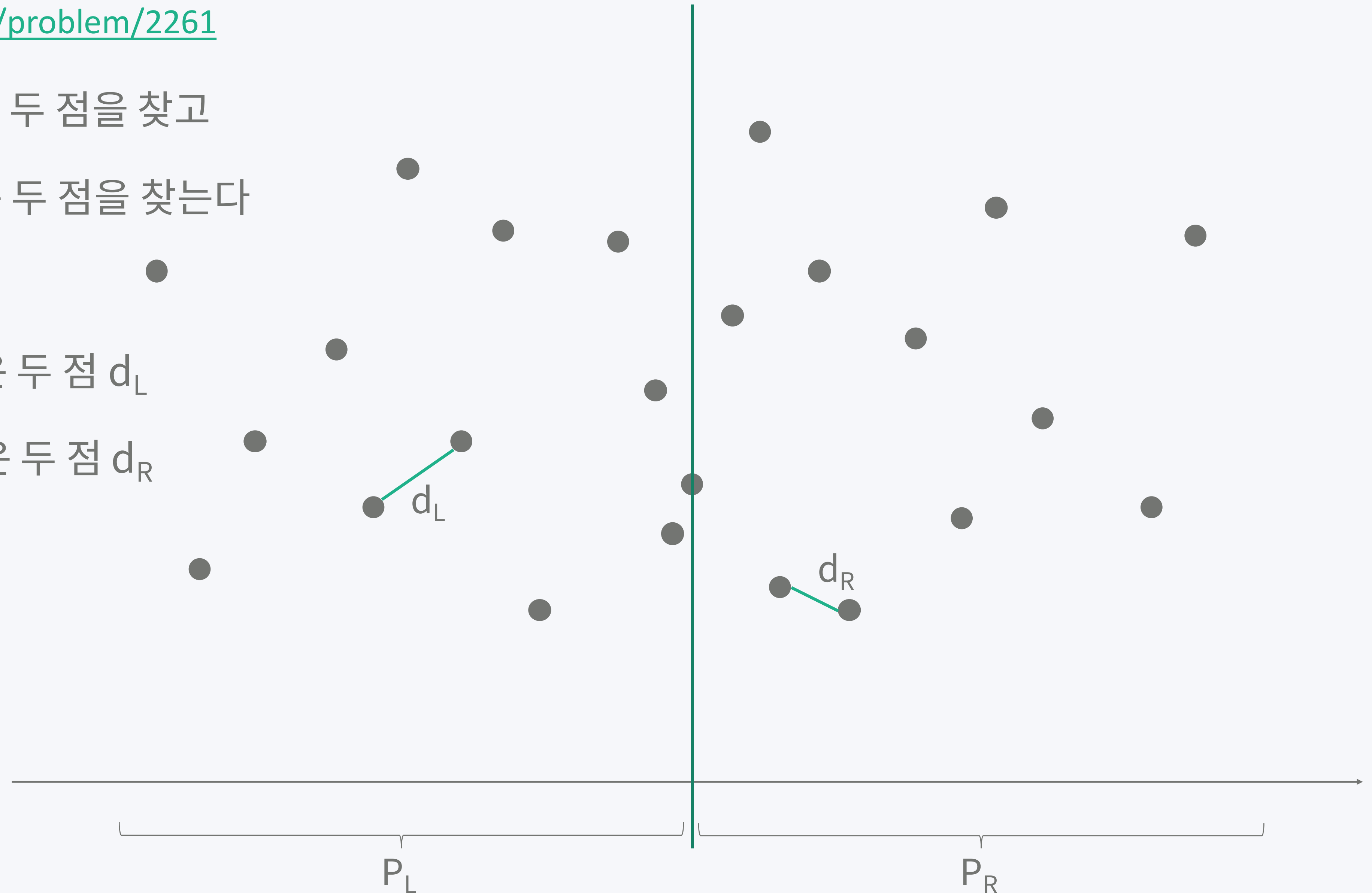
# 가장 가까운 두 점

74

<https://www.acmicpc.net/problem/2261>

- $P_L$ 에서 가장 가까운 두 점을 찾고
- $P_R$ 에서 가장 가까운 두 점을 찾는다

- $P_L$ 에서 가장 가까운 두 점  $d_L$
- $P_R$ 에서 가장 가까운 두 점  $d_R$

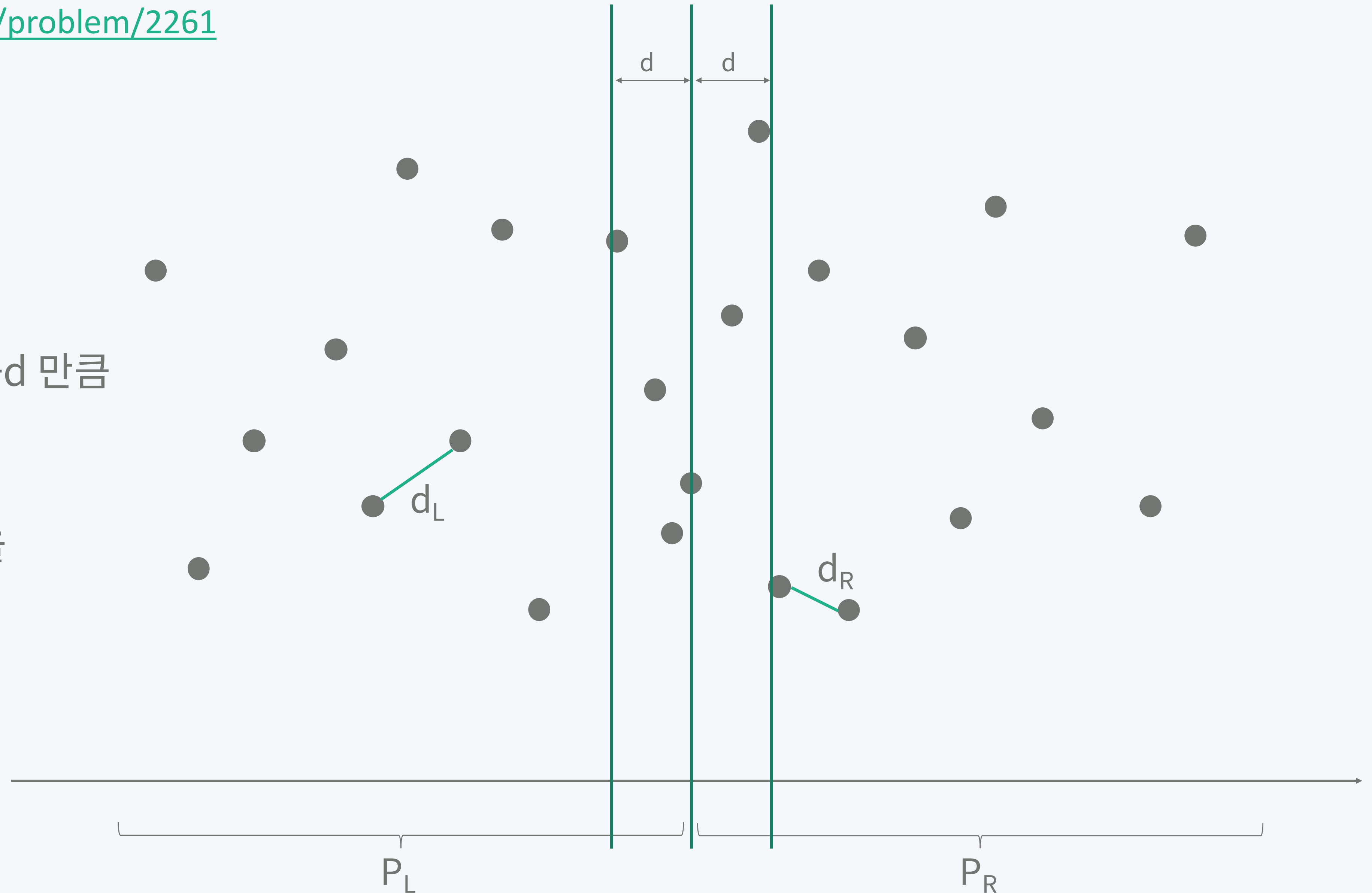


# 가장 가까운 두 점

75

<https://www.acmicpc.net/problem/2261>

- $d = \min(d_L, d_R)$
- 이라고 했을 때
- 가운데 점으로부터
- 가운데로부터  $-d, +d$  만큼
- 떨어진 곳에서
- 가장 가까운 두 점을
- 찾아야 한다

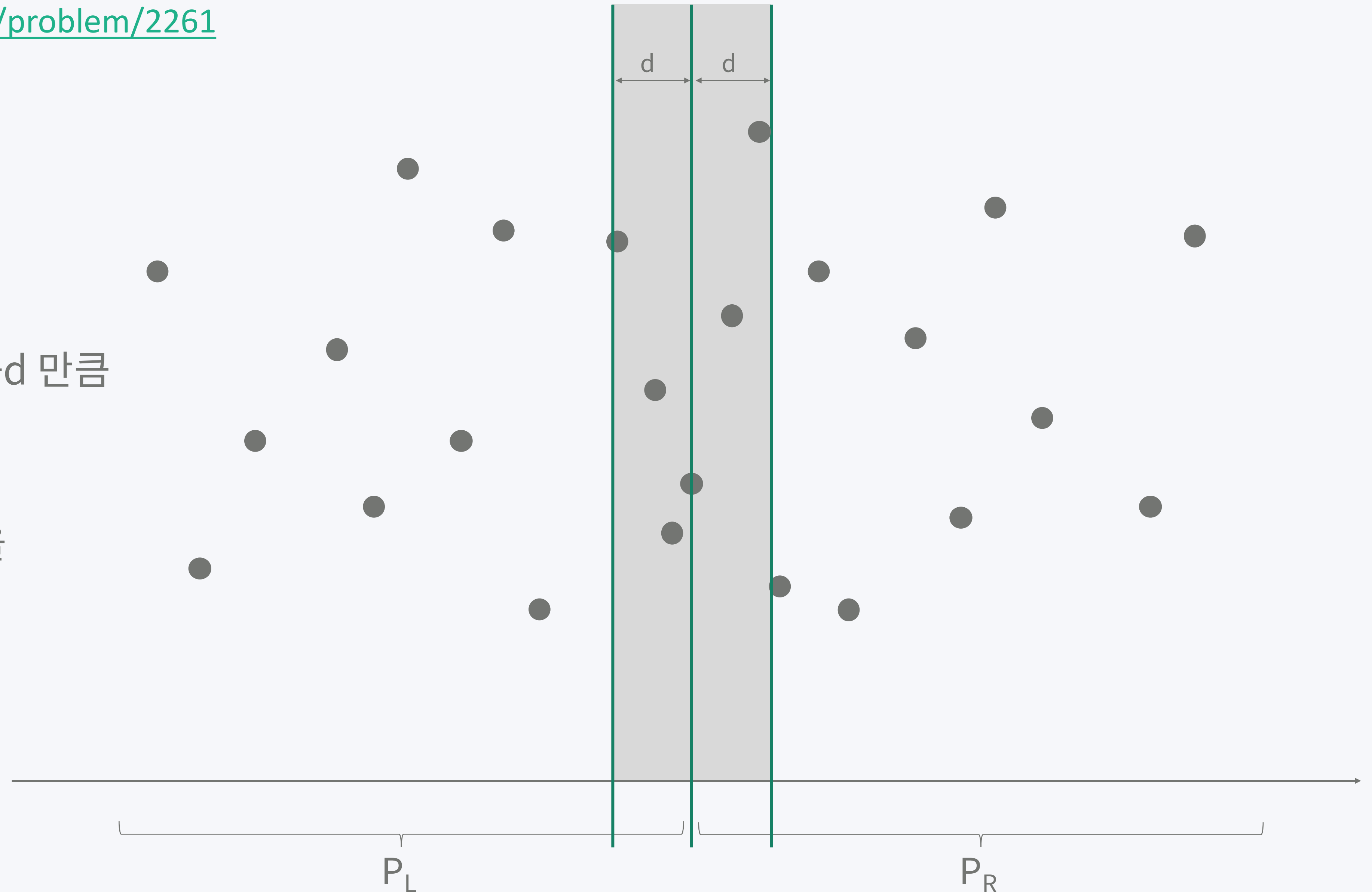


# 가장 가까운 두 점

76

<https://www.acmicpc.net/problem/2261>

- $d = \min(d_L, d_R)$
- 이라고 했을 때
- 가운데 점으로부터
- 가운데로부터  $-d, +d$  만큼
- 떨어진 곳에서
- 가장 가까운 두 점을
- 찾아야 한다

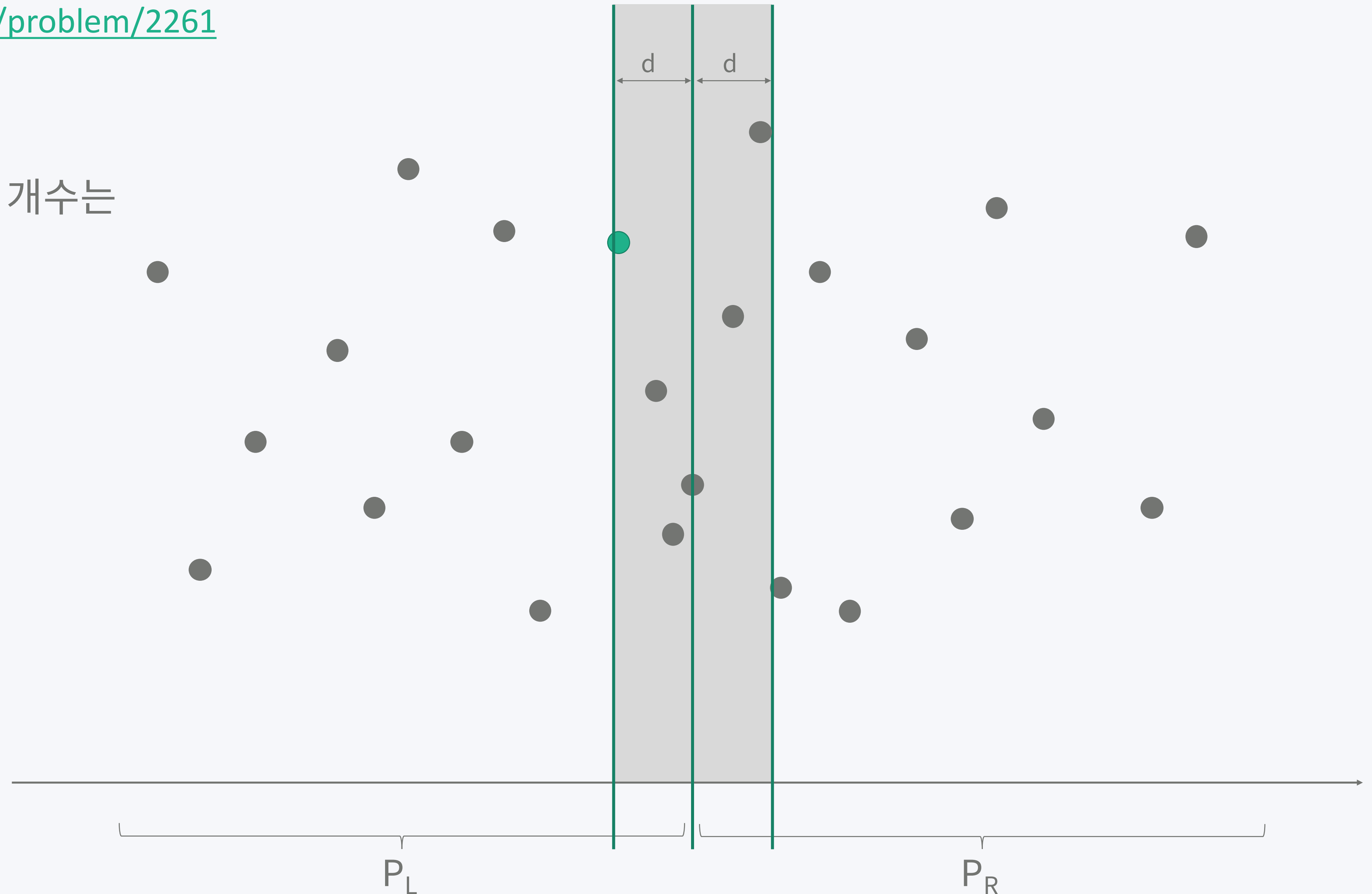


# 가장 가까운 두 점

77

<https://www.acmicpc.net/problem/2261>

- 한 점당
- 살펴봐야 하는 점의 개수는
- 6개이다.

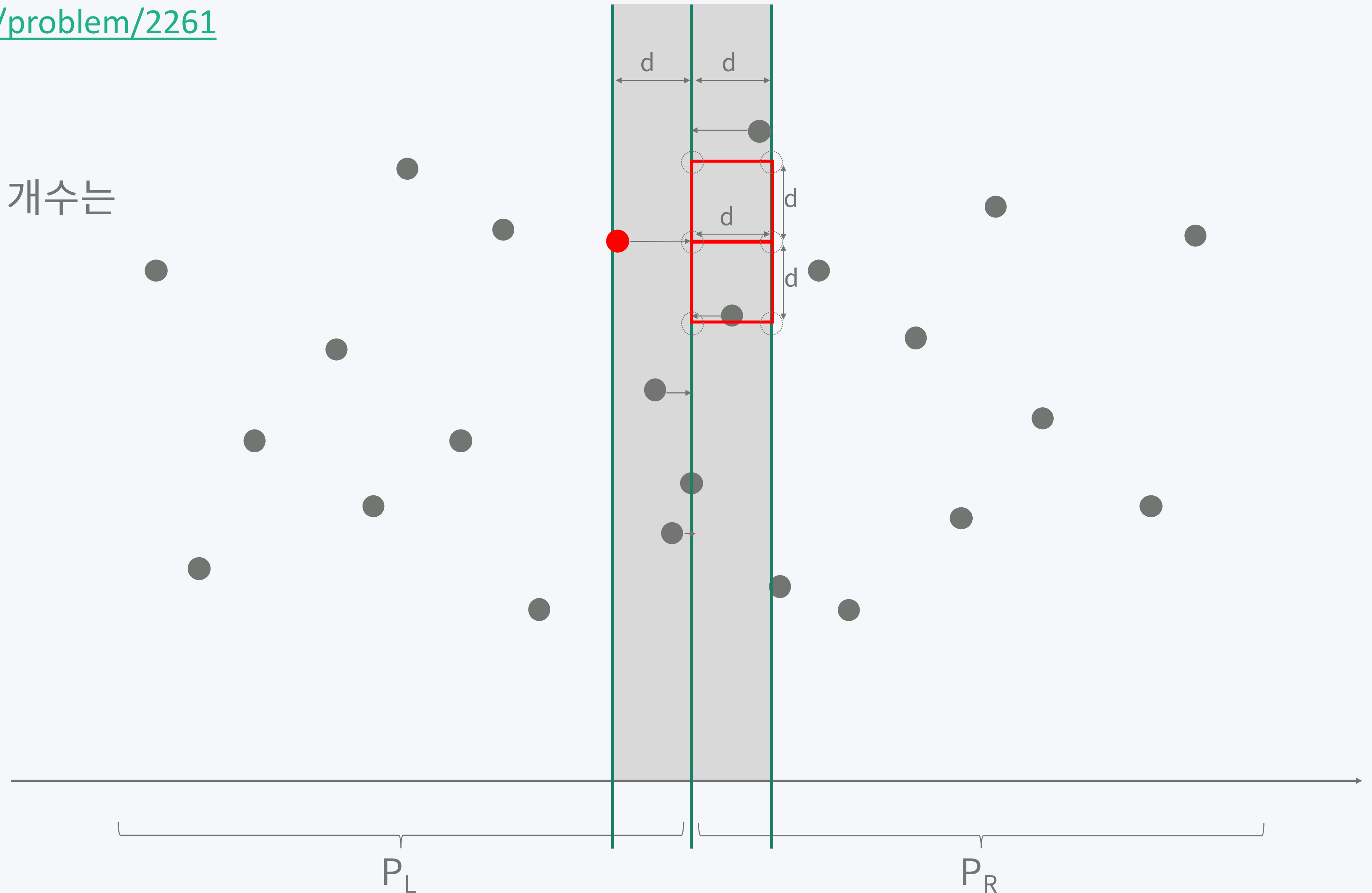


# 가장 가까운 두 점

78

<https://www.acmicpc.net/problem/2261>

- 한 점당
- 살펴봐야 하는 점의 개수는
- 6개이다.



# 가장 가까운 두 점

79

<https://www.acmicpc.net/problem/2261>

- C++: <https://gist.github.com/Baekjoon/4d240e06d2ec28fc8b78>