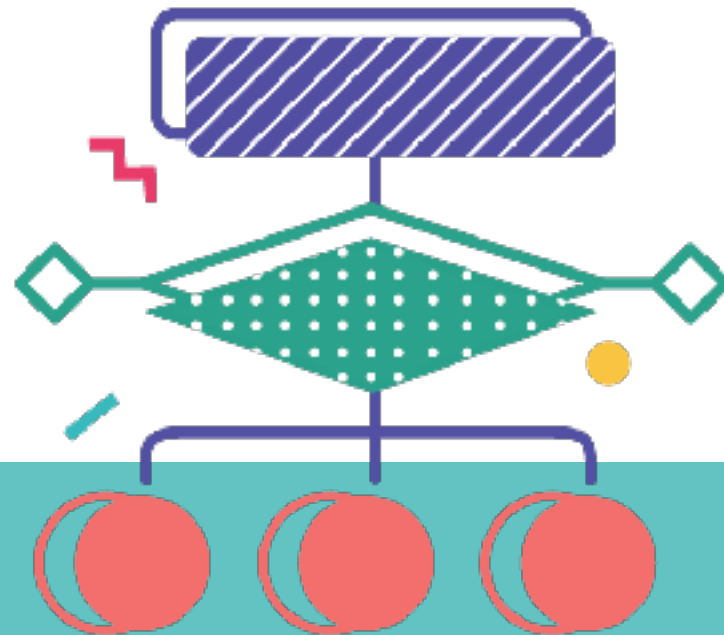


/* 알고리즘 I */

신현규 강사, 월 20:00

6월 26일 ~ 7월 23일



/* elice */

목차

- 00 지난시간 요약 및 숙제 풀이
- 01 Complexity Theory 맛보기
- 02 재귀호출을 이용한 문제 해결
- 03 분할정복법 (Divide & Conquer)

00 지난시간 요약 및 숙제 풀이

목표 수립

수강 목적 및 목표를 충분히 구체화 함

내가 이 알고리즘 강의를 왜 수강하는가 ?

수강생의 입장에서 나의 목표는 무엇인가 ?

모든 수업 시간에 참석한다 (O)

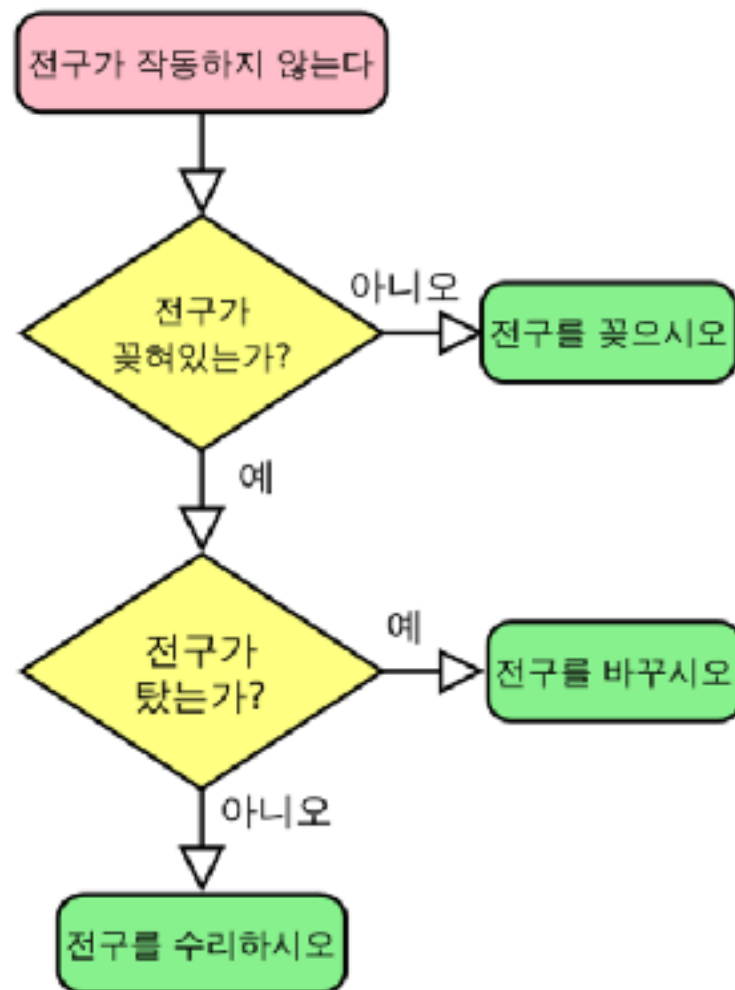
제공되는 모든 문제를 100점으로 해결한다 (O)

매 강의 시간마다 채팅창에 한 단어라도 남긴다 (O)

수업 시간에 진행되는 코딩을 모두 따라서 해본다 (O)

알고리즘

문제를 해결하는 방법



```
def fixBulb(bulb) :  
    if not bulb.isEmpty() :  
        bulb.create()  
  
    elif bulb.isBurnt :  
        bulb.change()  
  
    else :  
        bulb.fix()
```

<https://ko.wikipedia.org/wiki/%EC%88%9C%EC%84%9C%EB%8F%84>

알고리즘 커리큘럼

1주차	과정 소개, 재귀호출
2주차	문제 해결의 절차, 완전탐색, 시간복잡도
3주차	분할정복법
4주차	탐욕적 기법
5주차	동적 계획법 I
6주차	동적 계획법 II
7주차	그래프 알고리즘 I
8주차	그래프 알고리즘 II, 강의 요약

문제 해결의 절차

1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 문제를 해결한다는 것을 증명한다
4. 알고리즘이 제한시간 내에 동작한다는 것을 보인다
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

[예제 1] k번째 숫자 찾기

n개의 숫자 중 “지금까지 입력된 숫자들 중에서 k번째 숫자”는 ?
(단, $1 \leq k \leq n \leq 100$)

입력의 예

```
10 3  
1 9 8 5 2 3 5 6 2 10
```

출력의 예

```
-1 -1 9 8 5 3 3 3 2 2
```


시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    sum = sum + i
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0
```

```
for i in range(n) :  
    sum = sum + i
```

n 개

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    sum = sum + i
```

O(n)

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    for j in range(n) :  
        sum = sum + i + j
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    for j in range(n) :  
        sum = sum + i + j
```

$O(n^2)$

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    for j in range(i) :  
        sum = sum + i + j
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
sum = 0  
  
for i in range(n) :  
    for j in range(i) :  
        sum = sum + i + j
```

$O(n^2)$

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :  
    for v in myList :  
        if v == target :  
            return True  
  
    return False
```

시간복잡도 (Time Complexity)

알고리즘이 **대략** 몇개의 명령을 수행하는가?

프로그램의 수행 시간을 유추할 수 있음

```
def findNumber(myList, target) :  
    for v in myList :  
        if v == target :  
            return True  
  
    return False
```

O(n)

시간복잡도와 실제 수행 시간

많은 명령을 수행한다 = 오래 걸린다

시간복잡도와 실제 수행 시간

많은 명령을 수행한다 = 오래 걸린다

몇 개의 명령을 수행해야 1초가 걸리는가 ?

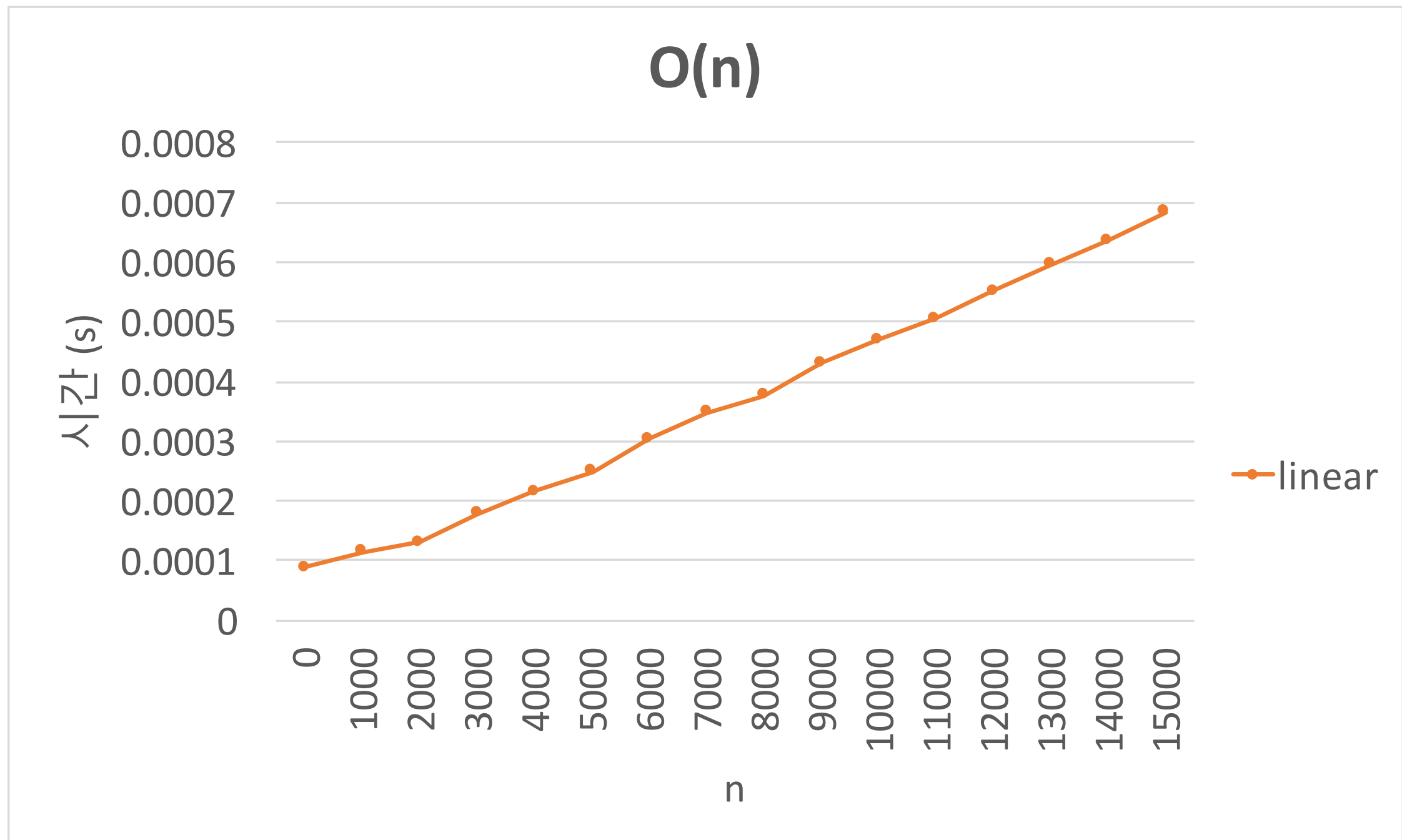
실험

아래 코드에 대하여 Elice에서 수행 시간을 측정

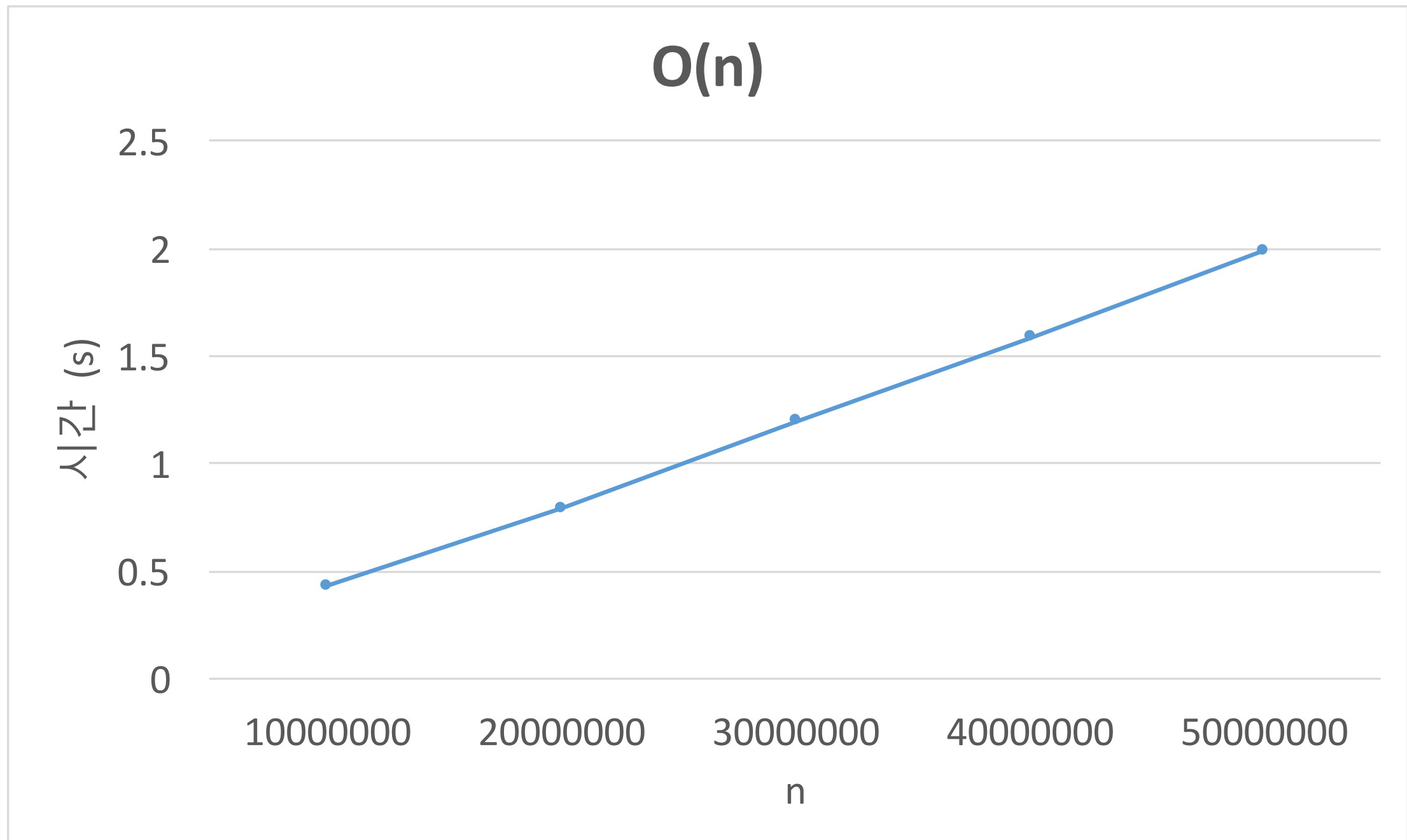
```
for i in range(n) :  
    sum = sum + 1
```

```
for i in range(n) :  
    for j in range(n) :  
        sum = sum + 1
```

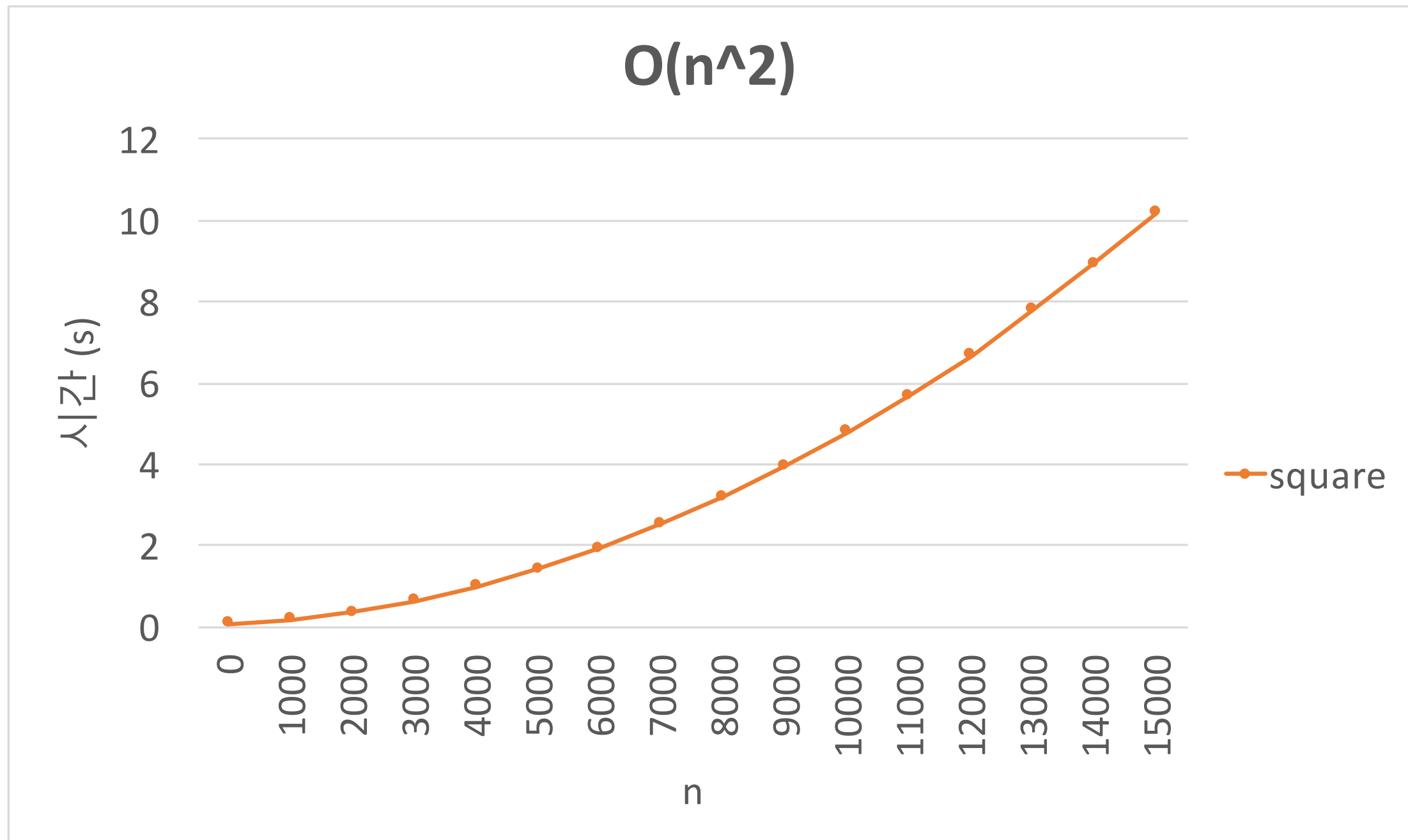
결과 : $O(n)$



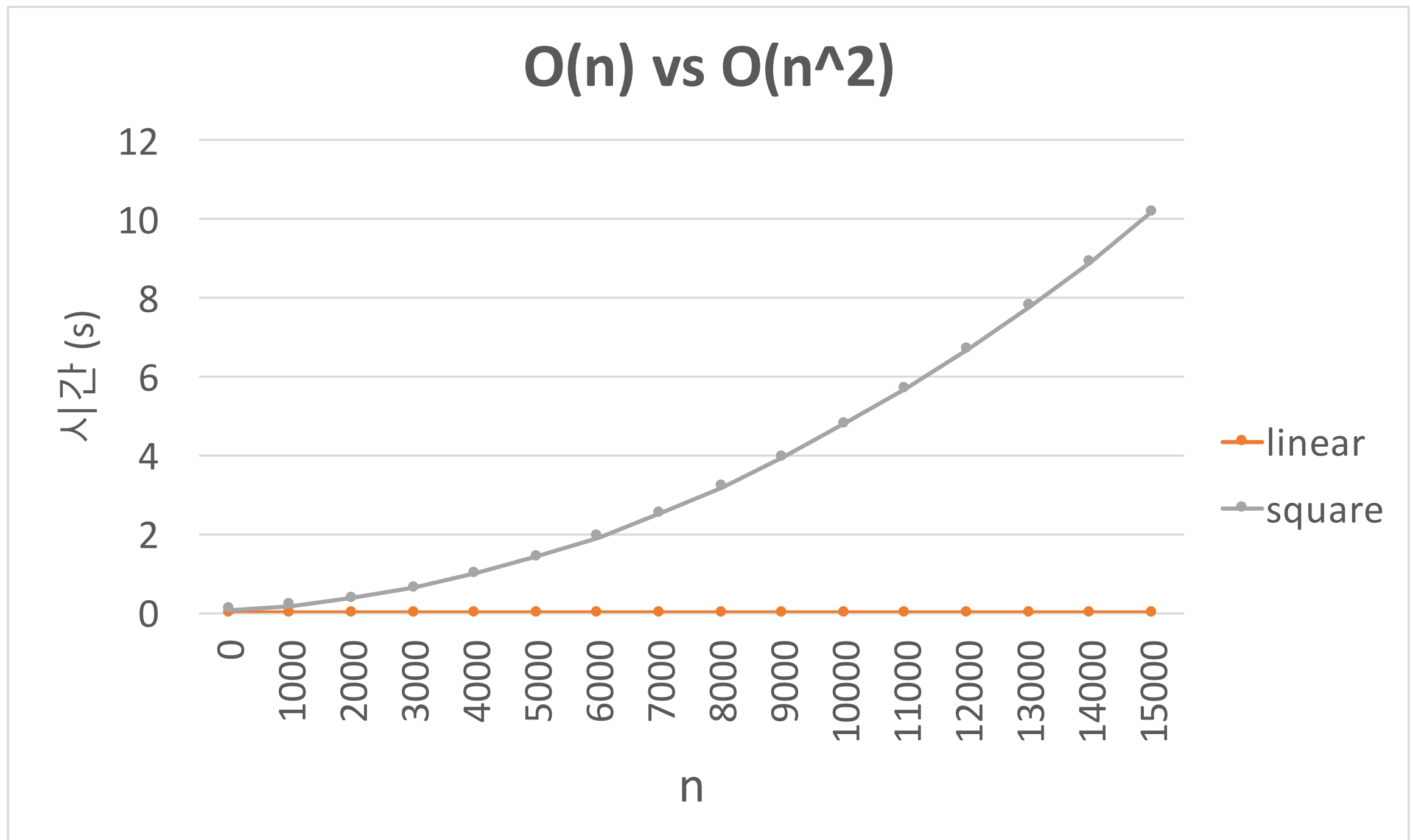
결과 : $O(n)$



결과 : $O(n^2)$



결과 : $O(n)$ vs $O(n^2)$



결론

대략 2500만개의 명령을
수행하면 1초가 걸린다

내 알고리즘이 최악의 경우에 2500만개를
수행하는지 고민해보자

완전 탐색 (Brute-Force)

가능한 모든 경우를 시도해 보는 것

완전 탐색 (Brute-Force)

가능한 모든 경우를 시도해 보는 것

가능한 모든 경우가 무엇인가 ?

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력
단, $1 \leq n \leq 100$

입력의 예

1 2 -4 5 3 -2 9 10

출력의 예

15

완전 탐색의 중요성

문제가 주어지면,

무.조.건.

완전 탐색법으로 먼저 시도해야 한다.

예외 없음. 안 해도 되는 경우 그런거 없음.

왜? 라고 물어볼 필요 없음. 그냥 무조건 해 봐야됨.

무조건. 그냥 무조건. 아니 그냥 무조건. 의문이 생겨선 안됨.

왜냐하면 의문이 생기기 전에 무조건 시도해 보았어야 하기 때문임.

상식적인 문제 해결의 흐름

A라는 방법으로 시도해봤더니 안되더라.

왜냐하면 a라는 이유 때문에.

따라서 a를 해결하는 B라는 방법으로 해봤더니 안되더라.

왜냐하면 b라는 이유때문에.

...

따라서 f를 해결하는 G라는 방법으로 해봤더니 되더라!!!

상식적인 문제 해결의 흐름

완전 탐색

A라는 방법으로 시도해봤더니 안되더라.

왜냐하면 a라는 이유 때문에.

따라서 a를 해결하는 **B**라는 방법으로 해봤더니 안되더라.

왜냐하면 b라는 이유때문에.

...

따라서 f를 해결하는 **G**라는 방법으로 해봤더니 되더라!!!

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

완전 탐색 : 가능한 모든 연속 구간을 모두 고려하자

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

완전 탐색 : 가능한 모든 연속 구간을 모두 고려하자

가능한 모든 연속 구간이 몇개나 있나 ?

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

↑ ↑
start end

연속 부분은 (start, end) 로 정의됨

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

↑ ↑
start end

연속 부분은 (start, end) 로 정의됨

총 28개의 경우가 있음

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

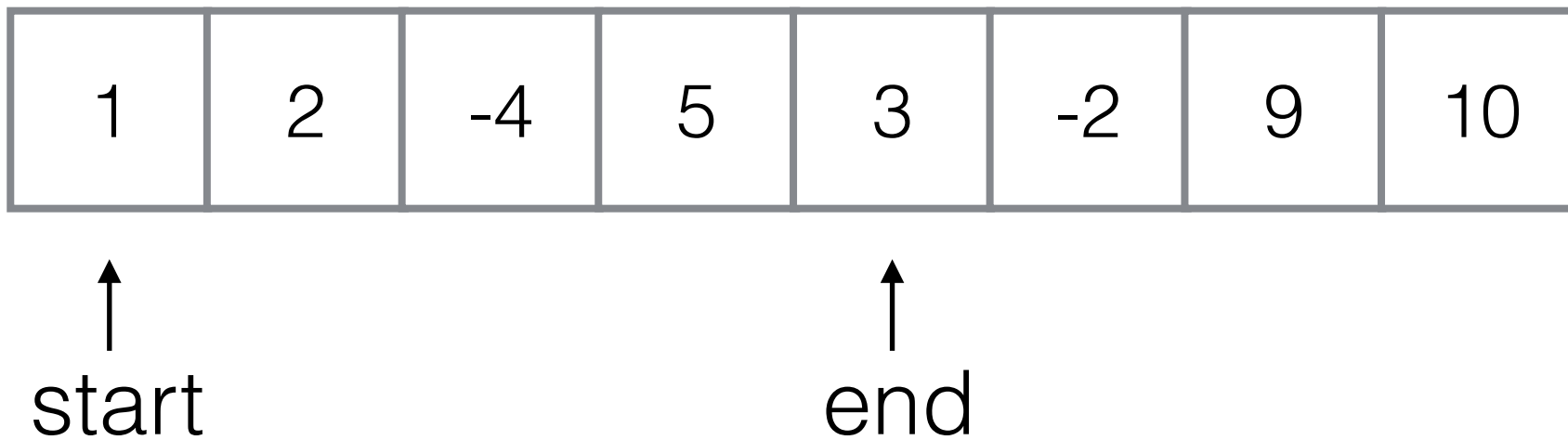
↑ ↑
start end

연속 부분은 (start, end) 로 정의됨

총 $O(n^2)$ 개의 경우가 있음

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력



한 개의 구간에 대하여 합을 구하는 데 걸리는 시간은 ?

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

↑
start

↑
end

한 개의 구간에 대하여 합을 구하는 데 걸리는 시간은 ?

O(n)

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력

1	2	-4	5	3	-2	9	10
---	---	----	---	---	----	---	----

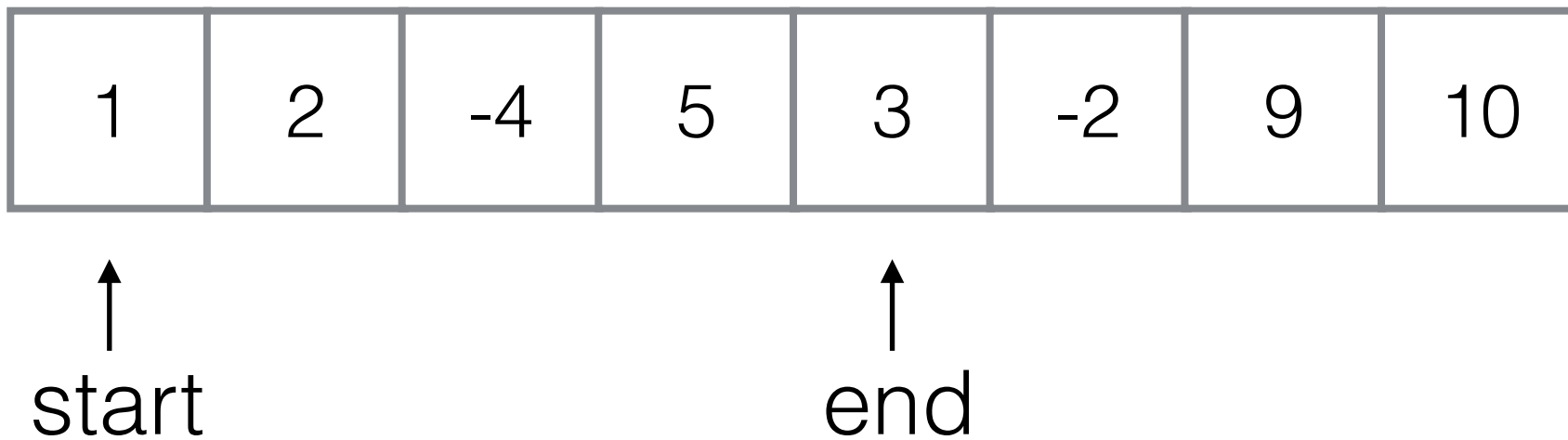
↑
start

↑
end

전체 시간 : (가능한 경우의 수) x (하나의 경우에 걸리는 시간)

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력



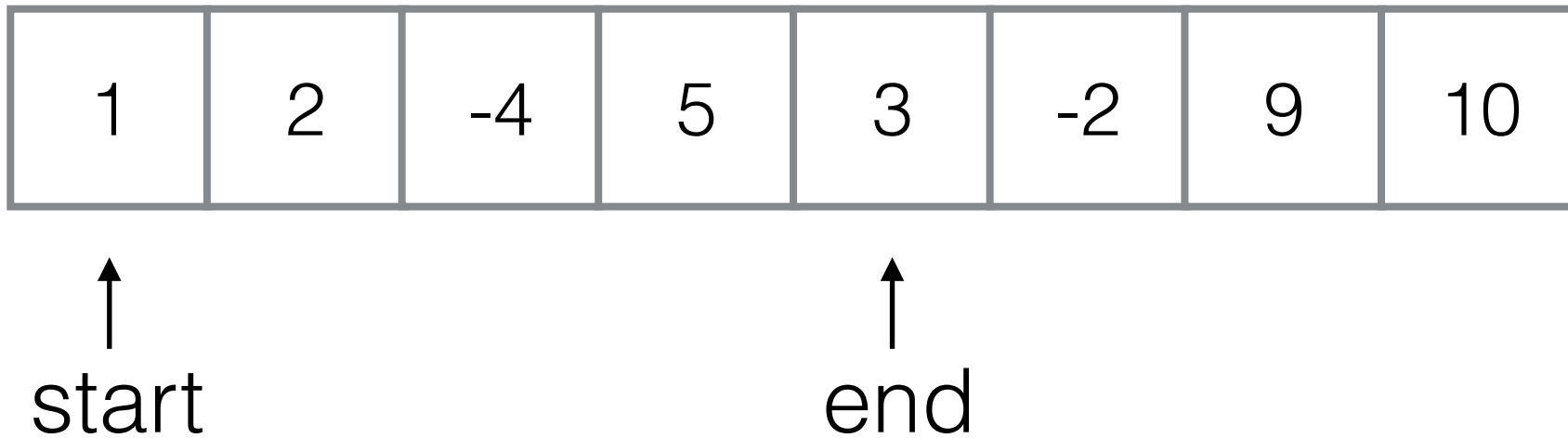
전체 시간 : (가능한 경우의 수) x (하나의 경우에 걸리는 시간)

O(n²)

O(n)

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력



전체 시간 : (가능한 경우의 수) x (하나의 경우에 걸리는 시간)

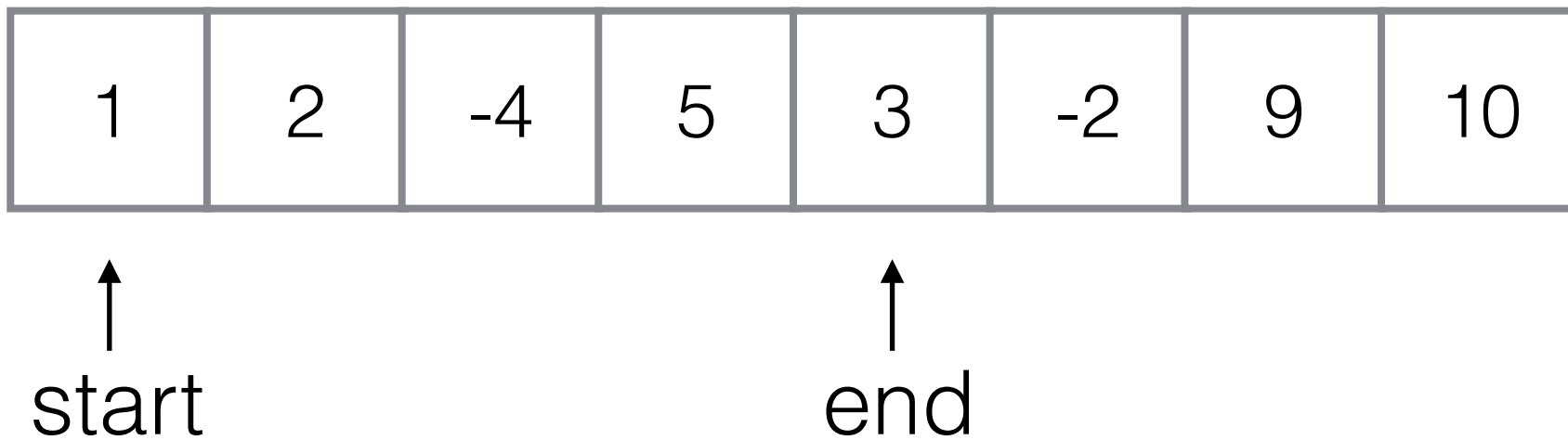
$O(n^3)$

O(n²)

O(n)

[예제 1] 연속 부분 최대합

연속된 부분을 선택하였을 때, 그 최대 합을 출력



전체 시간 : (가능한 경우의 수) x (하나의 경우에 걸리는 시간)

$O(n^3)$

O(n²)

O(n)

$n = 100$

문제 해결의 절차

1. 문제를 정확히 이해한다
2. 문제를 해결하는 알고리즘을 개발한다
3. 알고리즘이 문제를 해결한다는 것을 증명한다
4. 알고리즘이 제한시간 내에 동작한다는 것을 보인다
5. 알고리즘을 코드로 작성한다
6. 제출 후 만점을 받고 매우 기뻐한다

[예제 3] 균형 맞추기

n개의 숫자를 두 개의 그룹으로 나누어, 그 합을 가장 가깝게 하라
단, $1 \leq n \leq 10$

입력의 예

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

출력의 예

```
1
```


[예제 2] 멱집합 구하기

n개의 원소를 가지는 집합의 멱집합 구하기
단, $1 \leq n \leq 10$

입력의 예

```
3
```

출력의 예

```
1
1 2
1 2 3
1 3
2
2 3
3
```

[ADV 2] 뒤집기

한 칸을 뒤집으면 상,하,좌,우 색깔이 바뀜
최소로 뒤집는 횟수는 ?

단, $1 \leq n \leq 1000$, $1 \leq m \leq 1000$

입력의 예

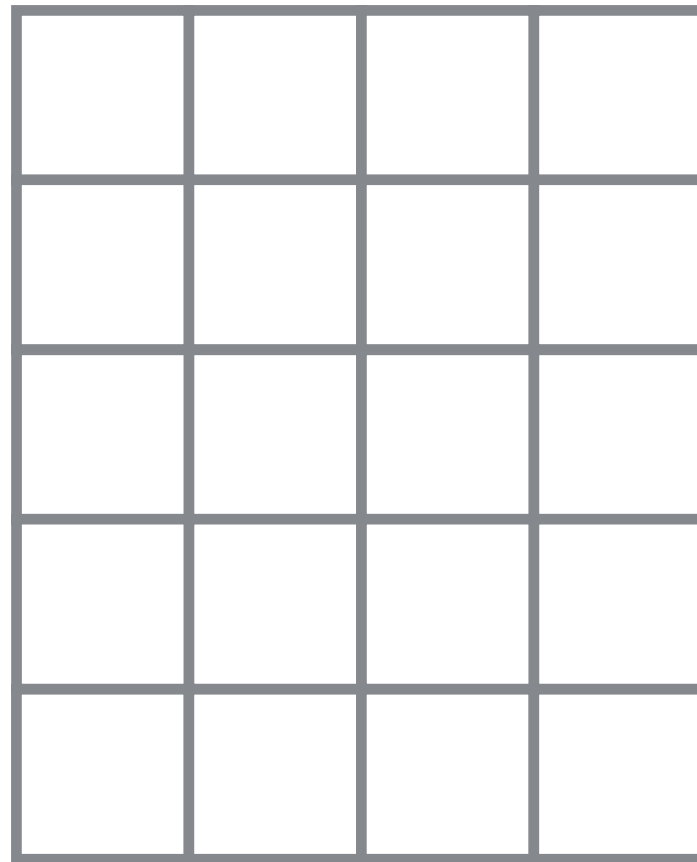
```
4 3
0 1 0
1 0 1
1 0 1
0 1 0
```

출력의 예

```
2
```

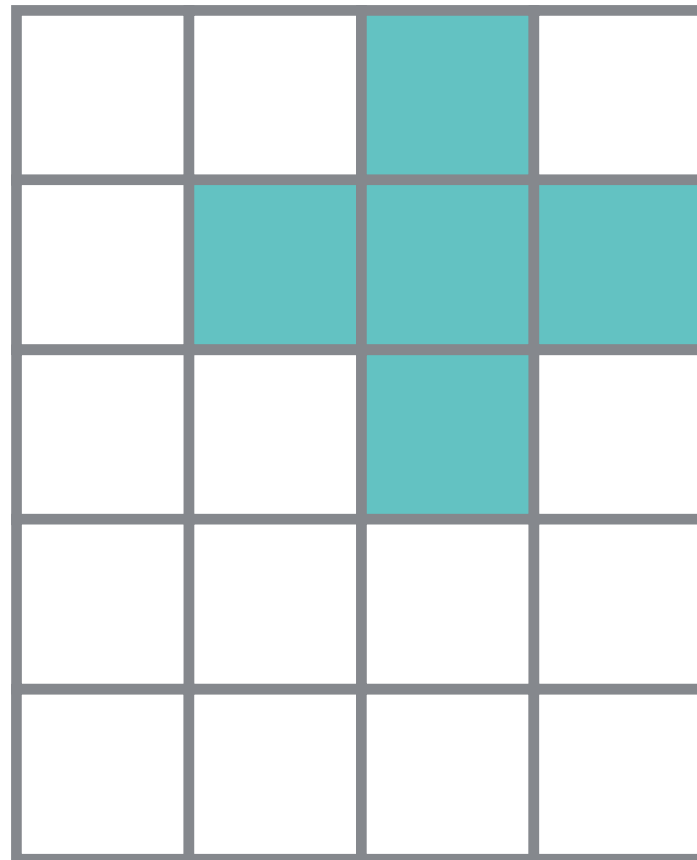
[ADV 2] 뒤집기

관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다



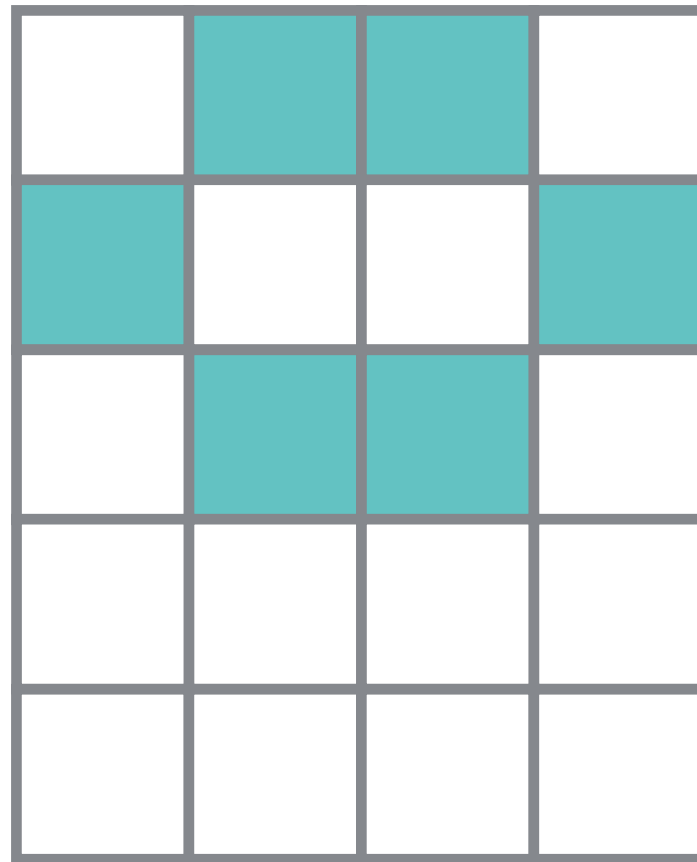
[ADV 2] 뒤집기

관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다



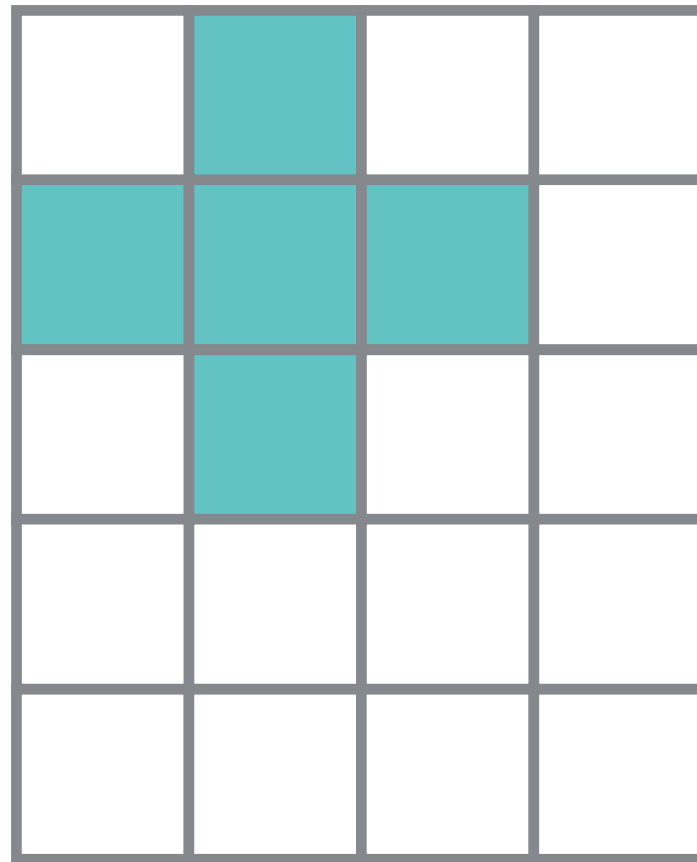
[ADV 2] 뒤집기

관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다



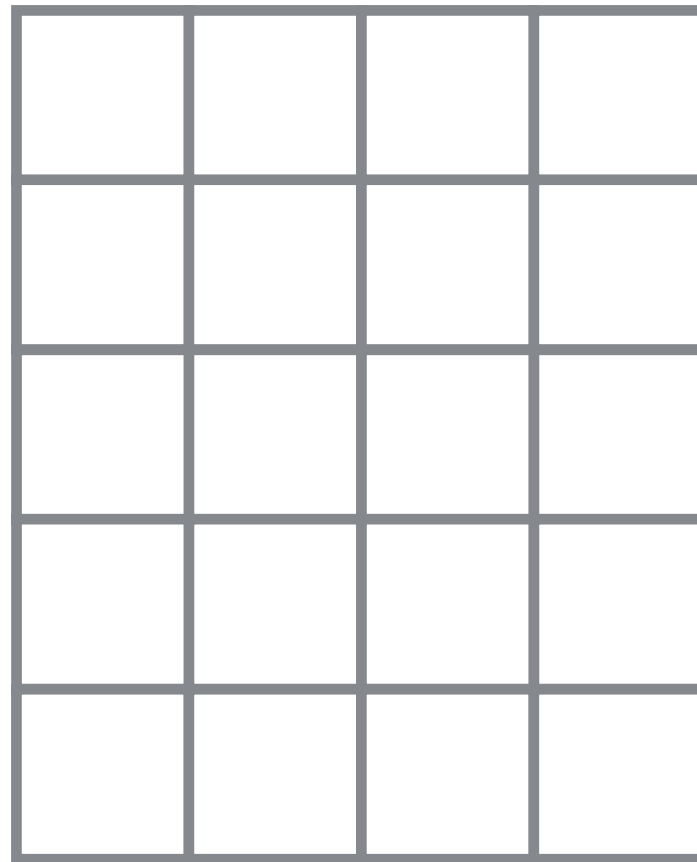
[ADV 2] 뒤집기

관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다



[ADV 2] 뒤집기

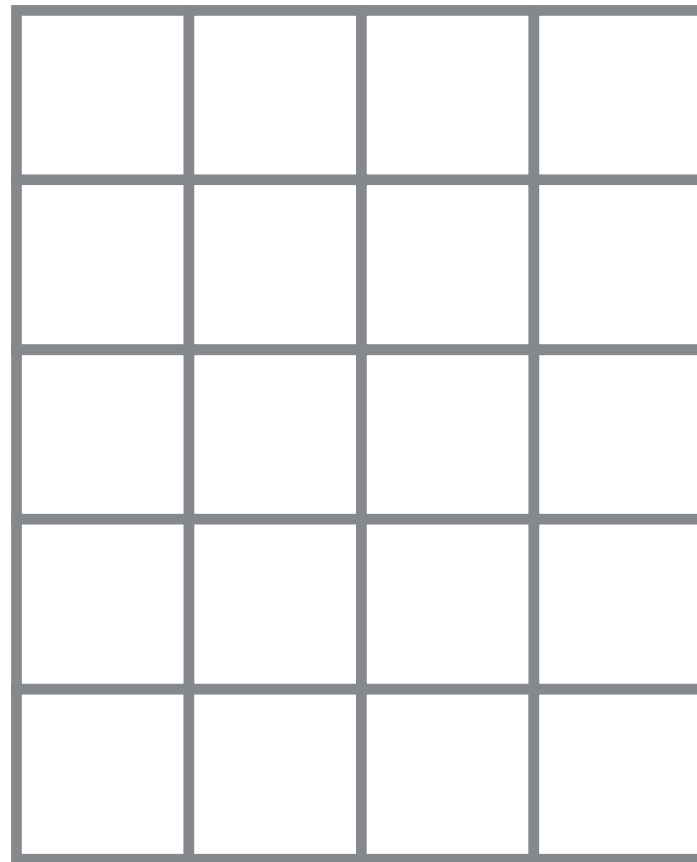
관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다



각 칸에 대하여 2가지 경우가 있음 : 누르거나, 누르지 않거나

[ADV 2] 뒤집기

관찰 1 : 한 칸을 두 번 뒤집을 필요가 없다

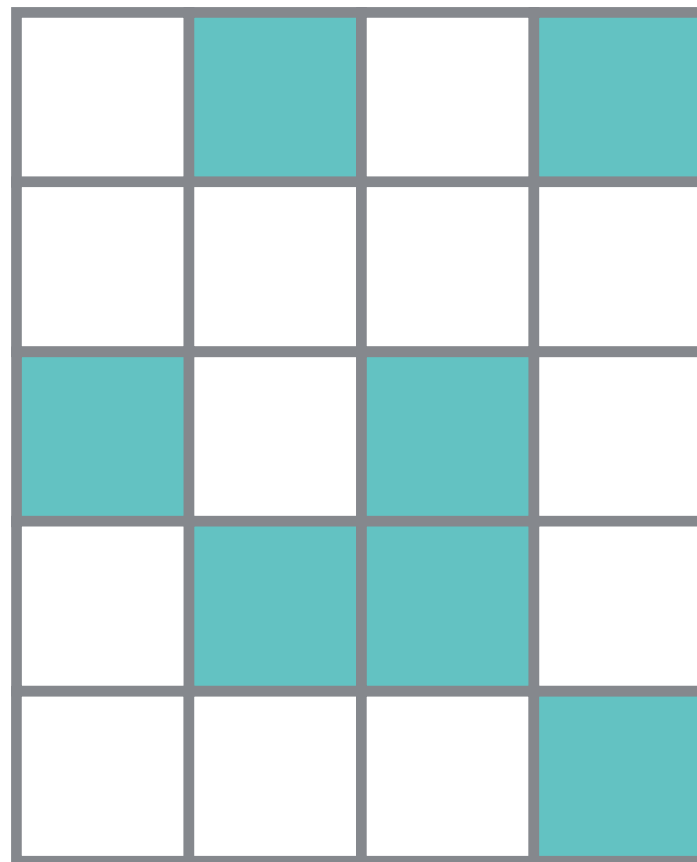


각 칸에 대하여 2가지 경우가 있음 : 누르거나, 누르지 않거나

$O(2^{NM})$ 가지의 경우가 존재 : 너무 많다

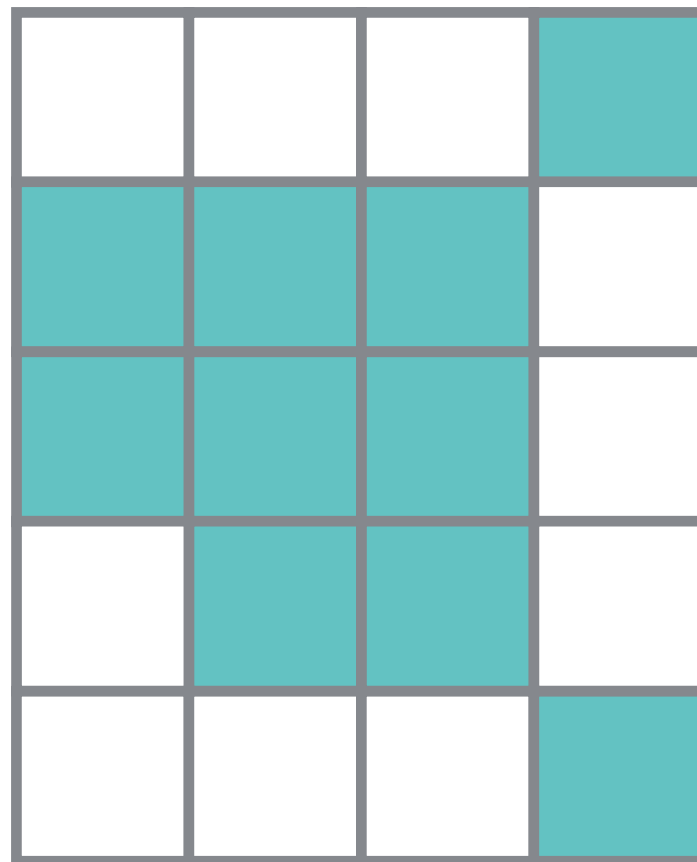
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



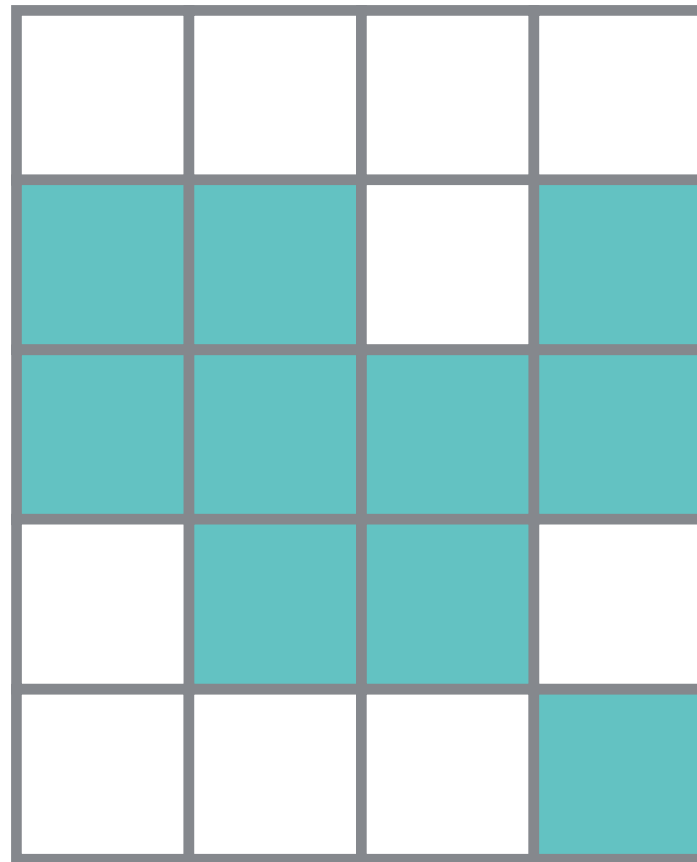
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



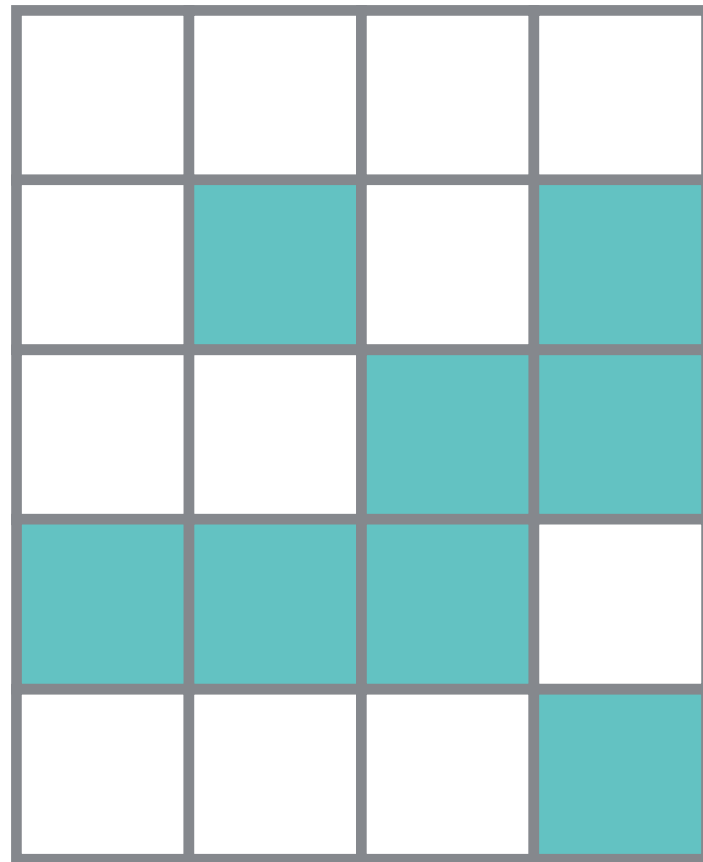
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



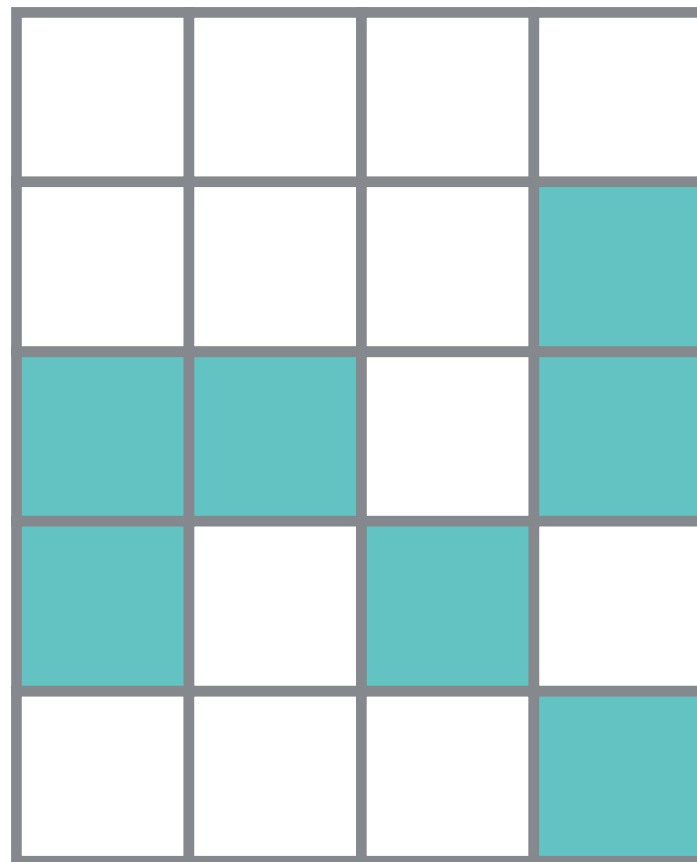
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



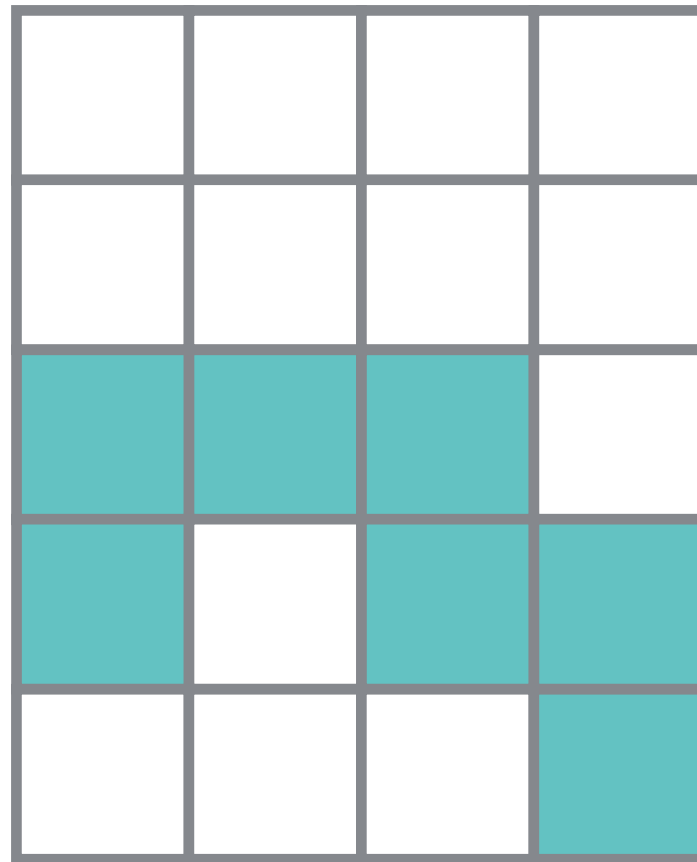
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



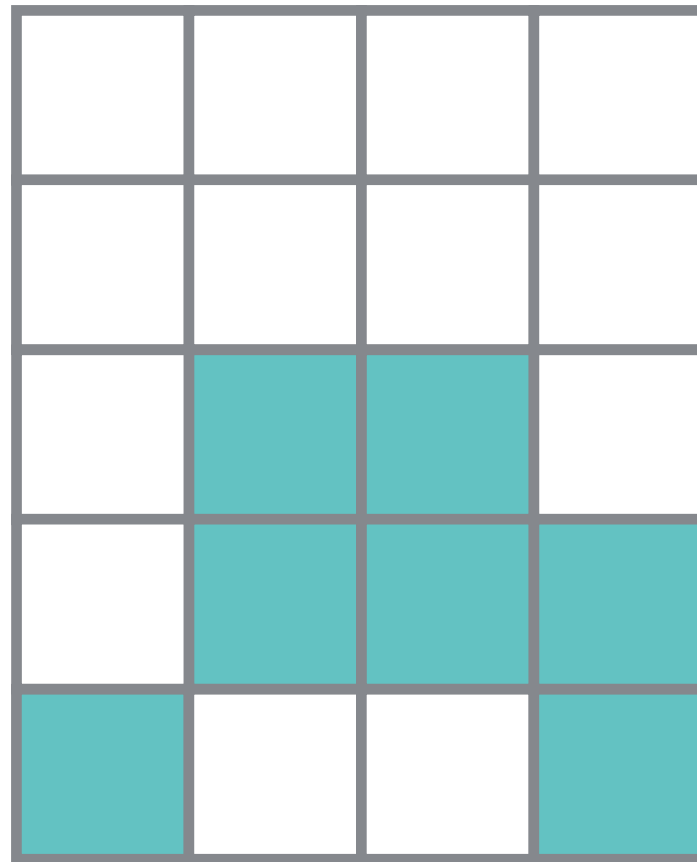
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



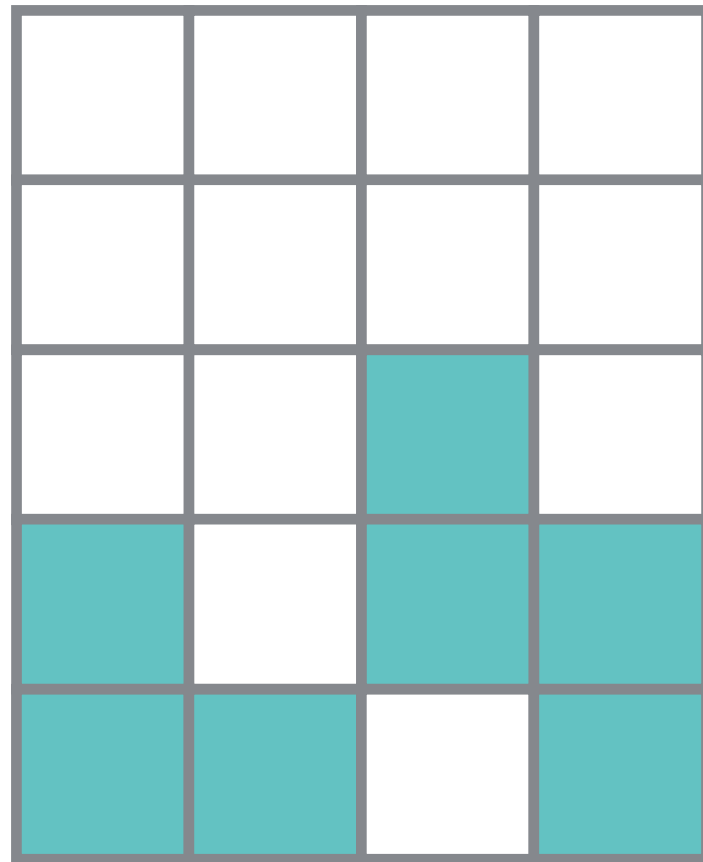
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



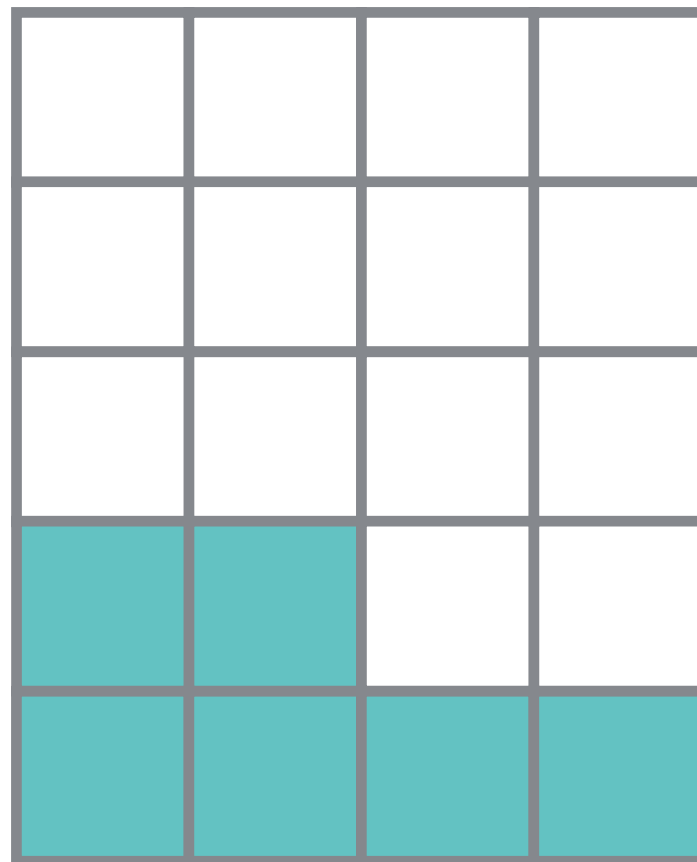
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



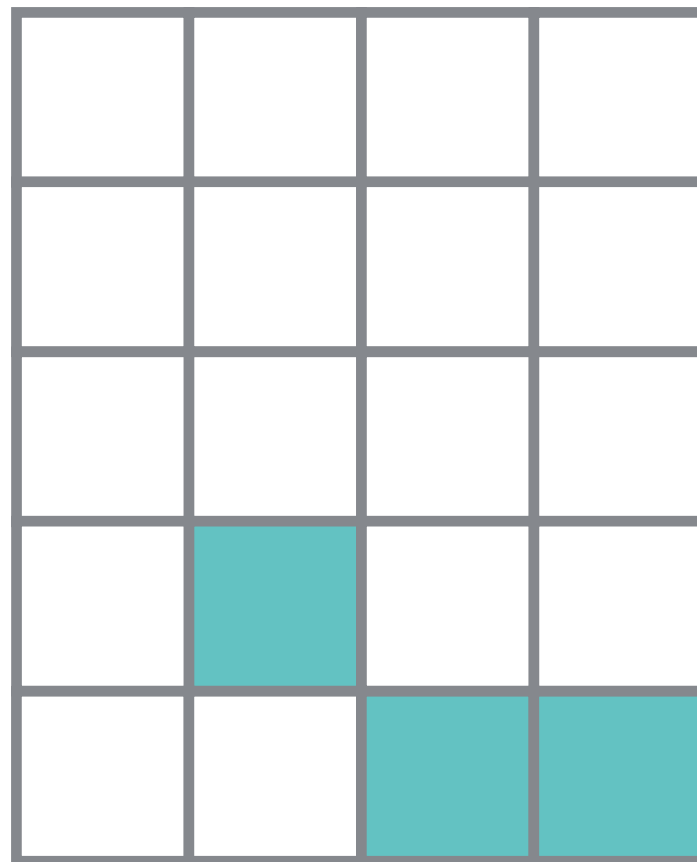
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



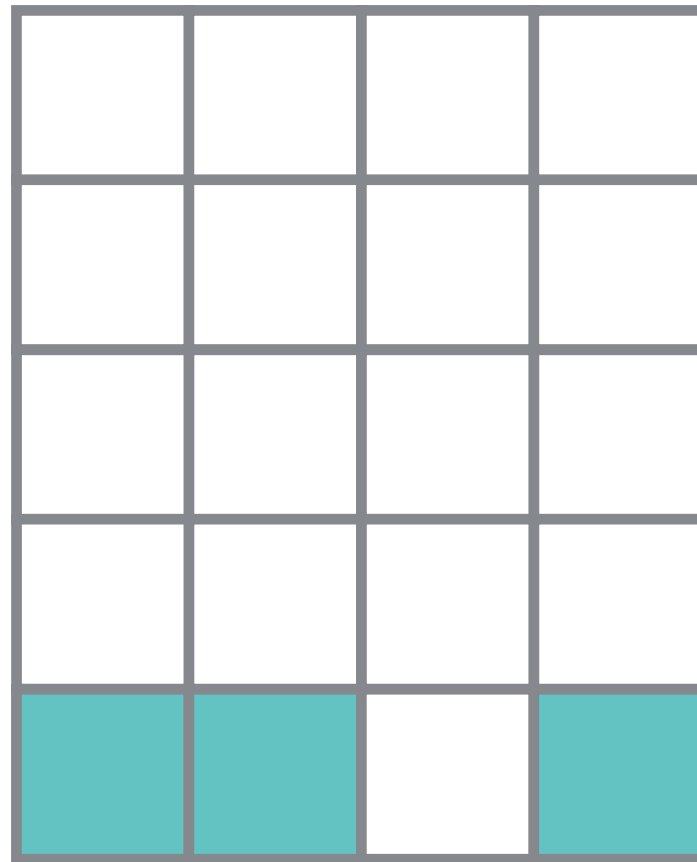
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



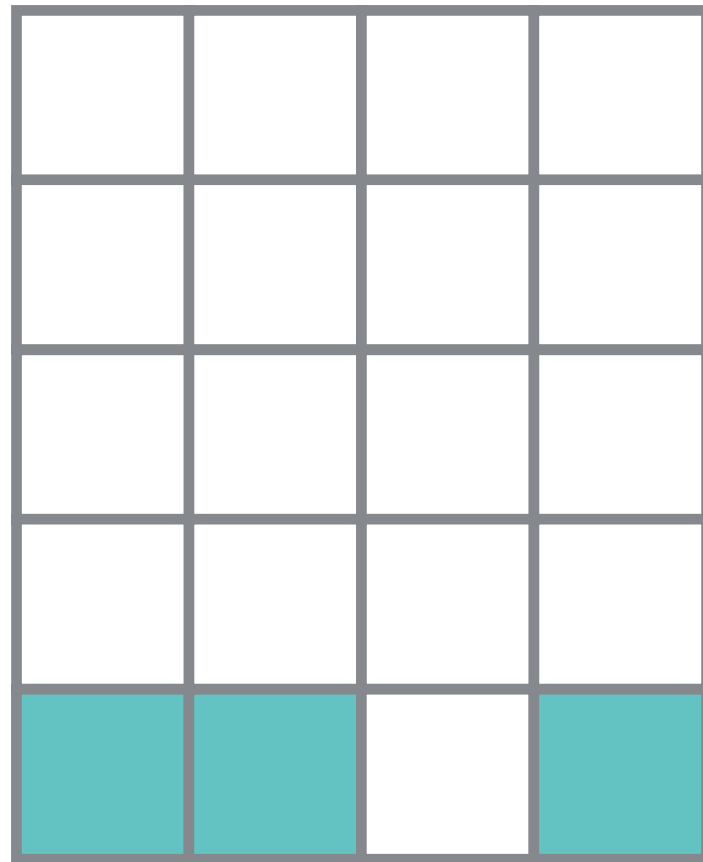
[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



[ADV 2] 뒤집기

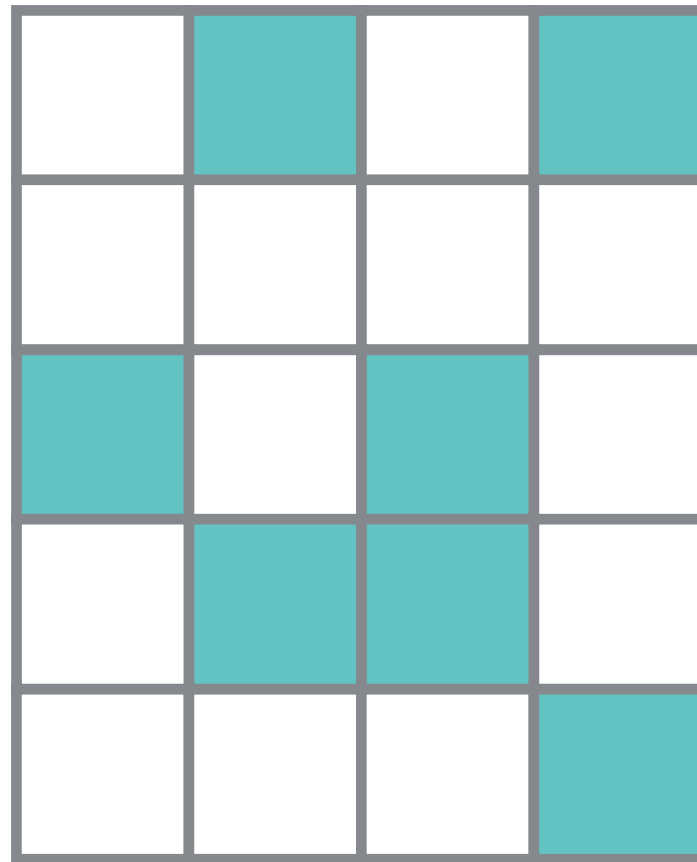
관찰 2 : + 모양임을 이용할 수 있다



IMPOSSIBLE

[ADV 2] 뒤집기

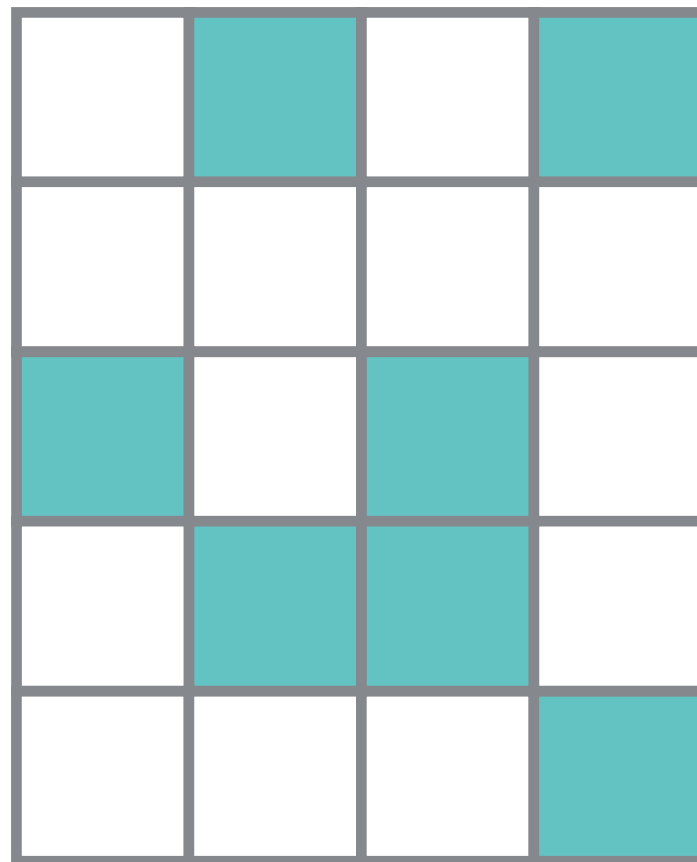
관찰 2 : + 모양임을 이용할 수 있다



첫 번째 줄만 모든 경우를 눌러보면, 두 번째 줄부터는 정해져있음

[ADV 2] 뒤집기

관찰 2 : + 모양임을 이용할 수 있다



$O(2^M) \times O(NM)$

01 Complexity Theory 맛보기

생각해볼 점

각 문제마다 풀이의 시간복잡도가 다르다

연속부분 최대합: $O(n^3)$, 균형 맞추기: $O(2^n)$

생각해볼 점

각 문제마다 풀이의 시간복잡도가 다르다

연속부분 최대합: $O(n^3)$, 균형 맞추기: $O(2^n)$

내 풀이가 얼마나 좋은 풀이인가 ?

예를 들어, 균형 맞추기는 $O(n^3)$ 에 불가능한가 ?

생각해볼 점

각 문제마다 풀이의 시간복잡도가 다르다

연속부분 최대합: $O(n^3)$, 균형 맞추기: $O(2^n)$

내 풀이가 얼마나 좋은 풀이인가 ?

예를 들어, 균형 맞추기는 $O(n^3)$ 에 불가능한가 ?

Complexity Theory

Complexity Theory

문제 자체에도 복잡도가 존재한다

빨리 풀 수 있는 문제가 있고, 그렇지 않은 문제가 있다

Complexity Theory

문제 자체에도 복잡도가 존재한다

빨리 풀 수 있는 문제가 있고, 그렇지 않은 문제가 있다

(정확하진 않지만 그렇게 틀린 말도 아닌 초 단순한 개념)

P class

다항 시간 내에 해결 가능한 **문제**들의 집합

NP-Complete class

다항 시간 내에 해결이 불가능한 **문제**들의 집합

알고리즘 과정에서 다루는 문제들

(거의 대부분) P 문제들만을 다룹니다

가능한 모든 경우는 2^n 개 처럼 보이지만 실제로는
적은 수의 경우만 고려하면 충분한 문제만을 다룹니다

알고리즘 과정에서 다루는 문제들

(거의 대부분) P 문제들만을 다룹니다

가능한 모든 경우는 2^n 개 처럼 보이지만 실제로는
적은 수의 경우만 고려하면 충분한 문제만을 다룹니다

알고리즘에서는 고려해야 하는 경우를 줄이는 방법을 배웁니다

어떻게 좀 더 줄일지에 대한 고민을 계속해서 하게 됩니다

알고리즘 과정에서 다루는 문제들

(거의 대부분) P 문제들만을 다룹니다

가능한 모든 경우는 2^n 개 처럼 보이지만 실제로는
적은 수의 경우만 고려하면 충분한 문제만을 다룹니다

알고리즘에서는 고려해야 하는 경우를 줄이는 방법을 배웁니다

어떻게 좀 더 줄일지에 대한 고민을 계속해서 하게 됩니다

하지만 대표적인 NP-Complete 문제는 알면 좋습니다

NP-Complete 로 유명한 문제를 굳이 다항시간에 풀려고 하는
일을 미연에 방지할 수 있기 때문입니다

02 재귀호출을 이용한 문제 해결

[예제 1] 가장 가까운 값 찾기

정렬된 n 개의 숫자 중 정수 m 과 가장 가까운 값을 찾아라
단, $1 \leq n \leq 100,000$

입력의 예

```
1 4 6 7 10 14 16  
8
```

출력의 예

```
7
```

이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

14

이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

14



이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

14

이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

14



이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

5

이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

5

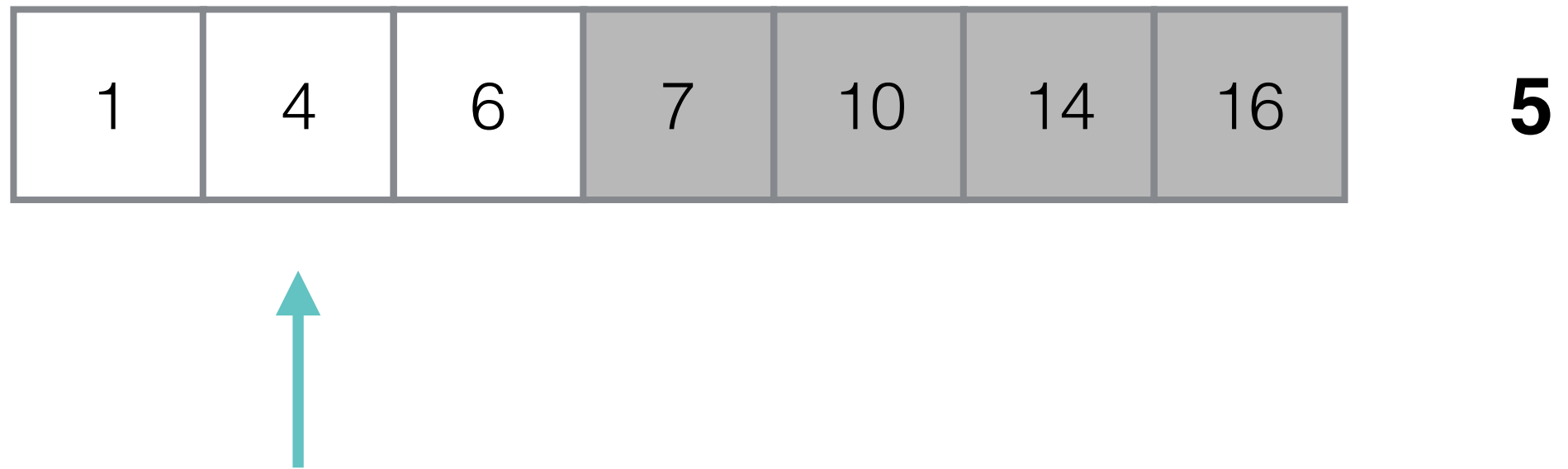


이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

5

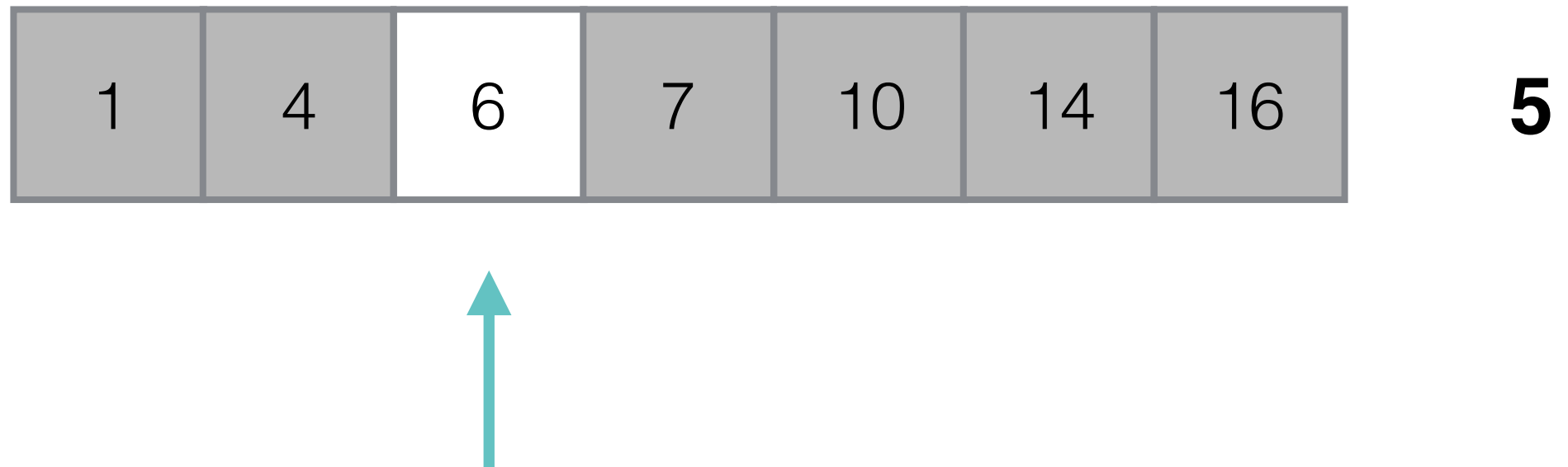
이진 탐색 (Binary search)



이진 탐색 (Binary search)

1	4	6	7	10	14	16	5
---	---	---	---	----	----	----	---

이진 탐색 (Binary search)



이진 탐색 (Binary search)

1	4	6	7	10	14	16
---	---	---	---	----	----	----

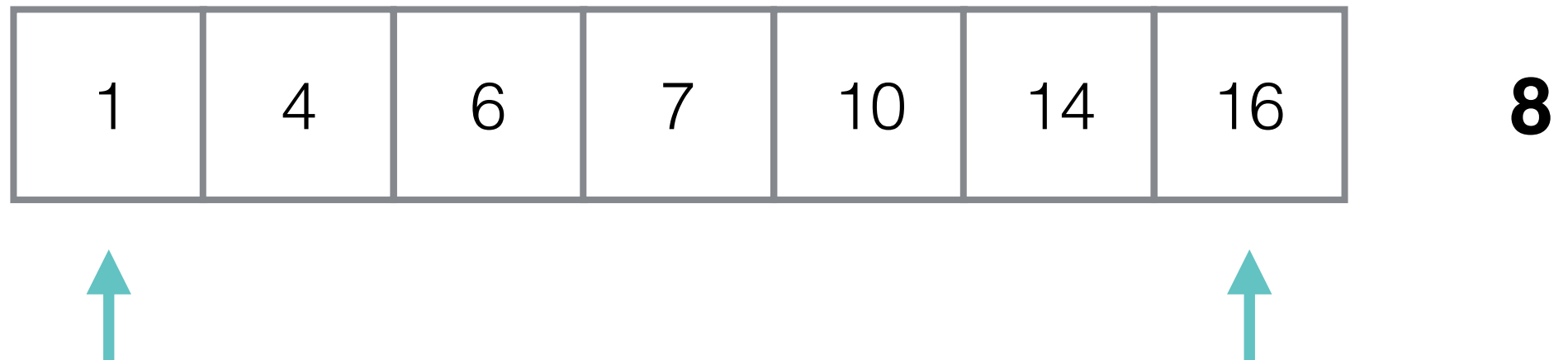
5

이진 탐색의 구현

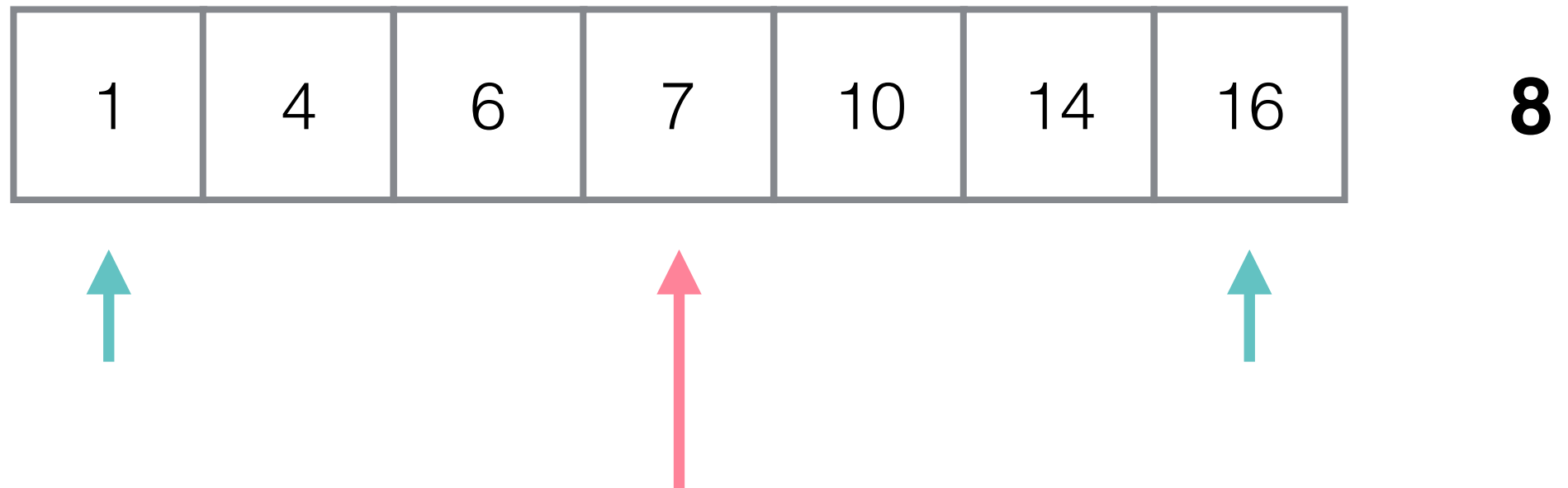
1	4	6	7	10	14	16
---	---	---	---	----	----	----

8

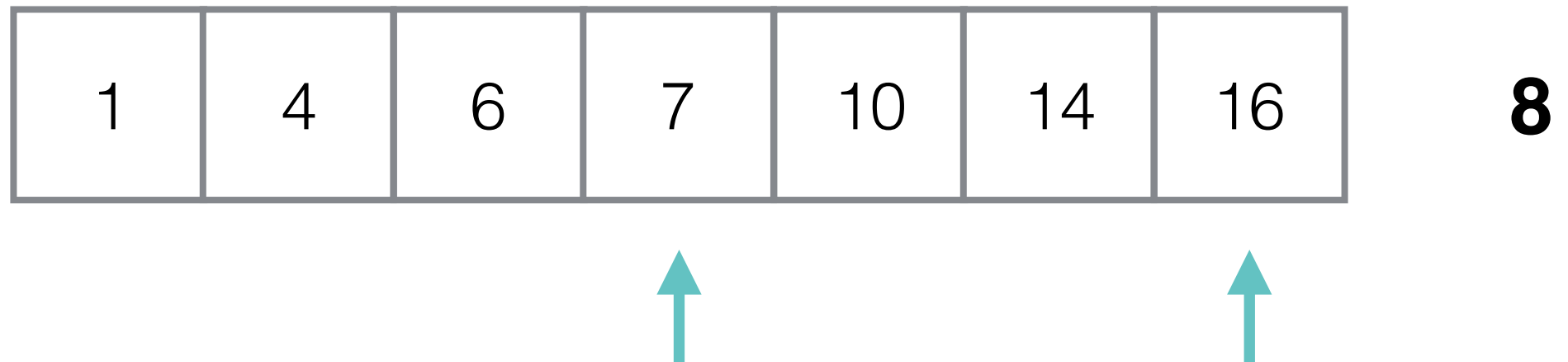
이진 탐색의 구현



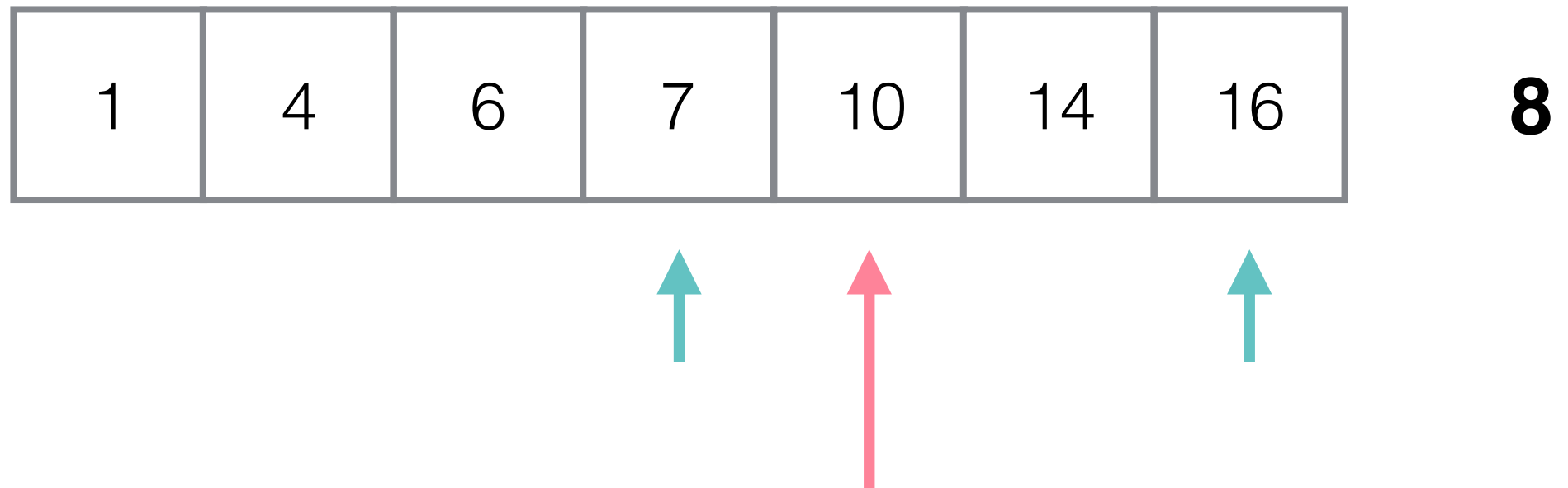
이진 탐색의 구현



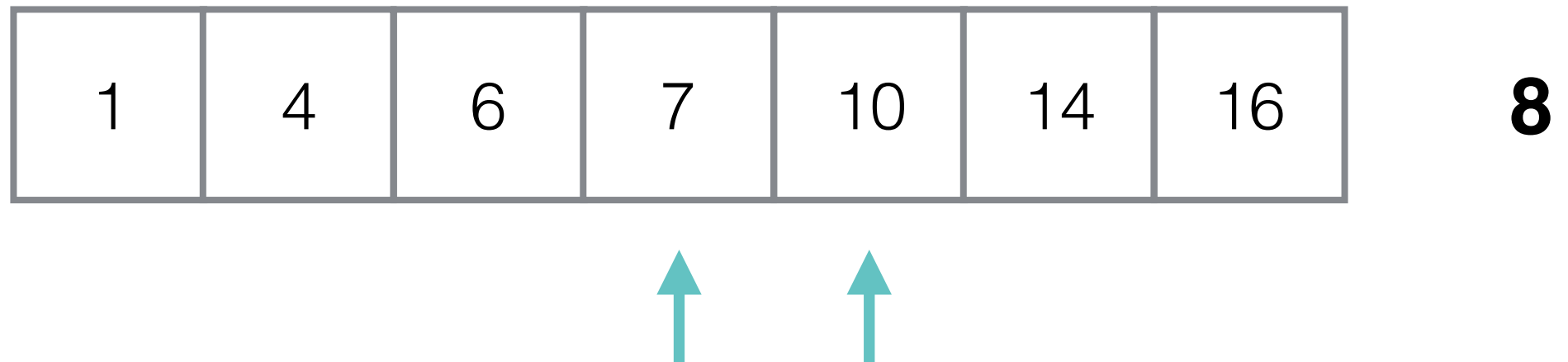
이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현

1	4	6	7	10	14	16
---	---	---	---	----	----	----

8



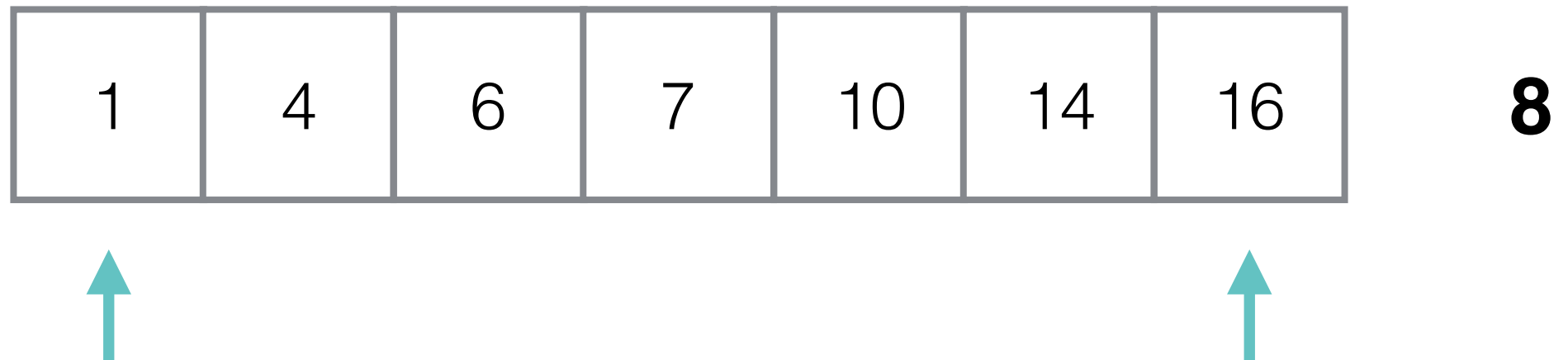
없음!

이진 탐색의 구현

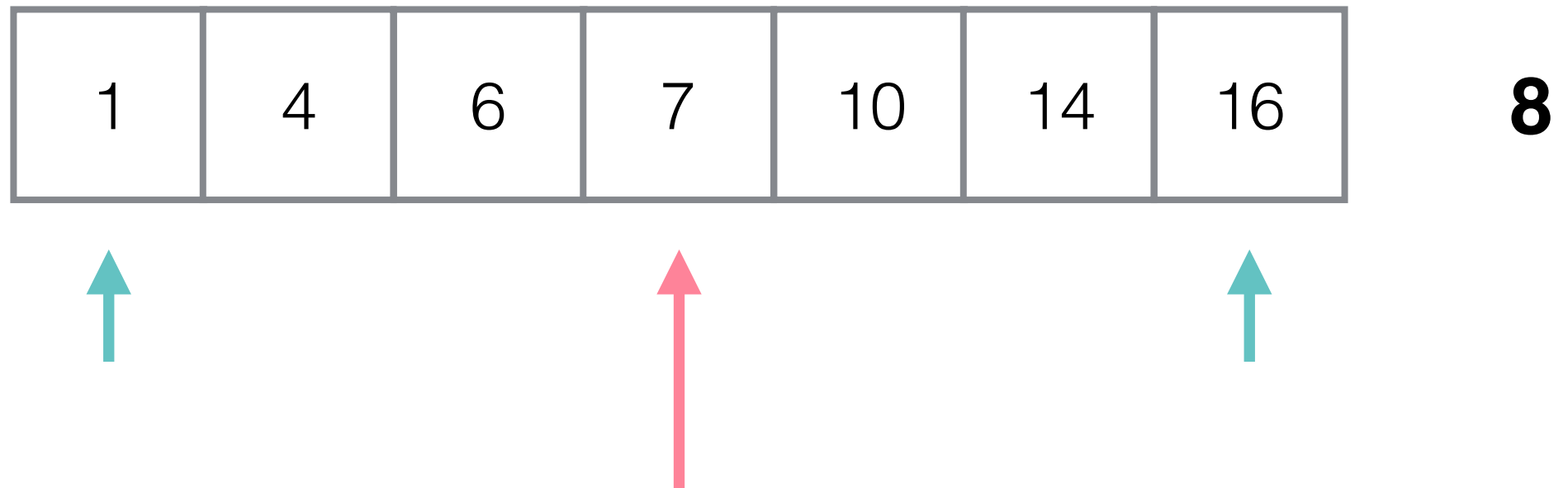
1	4	6	7	10	14	16
---	---	---	---	----	----	----

8

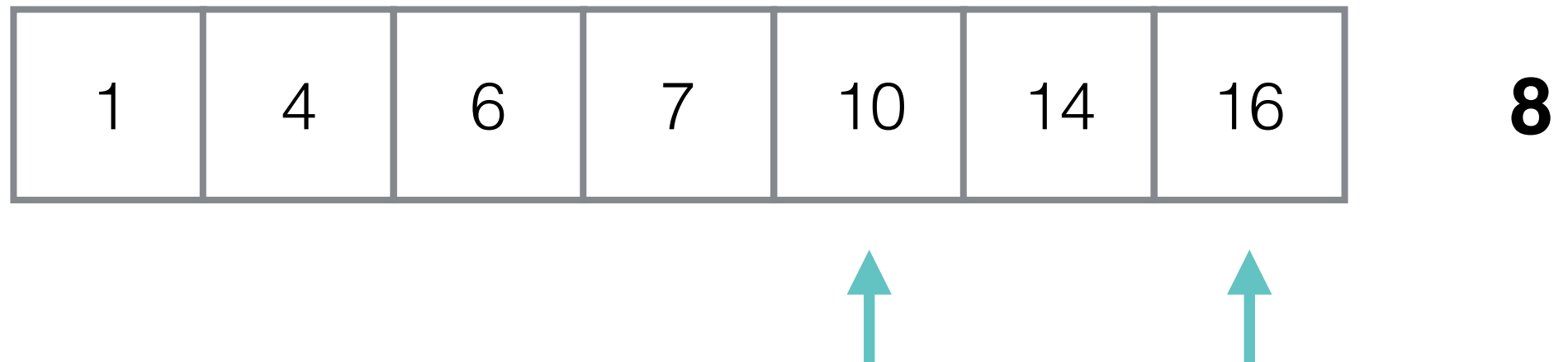
이진 탐색의 구현



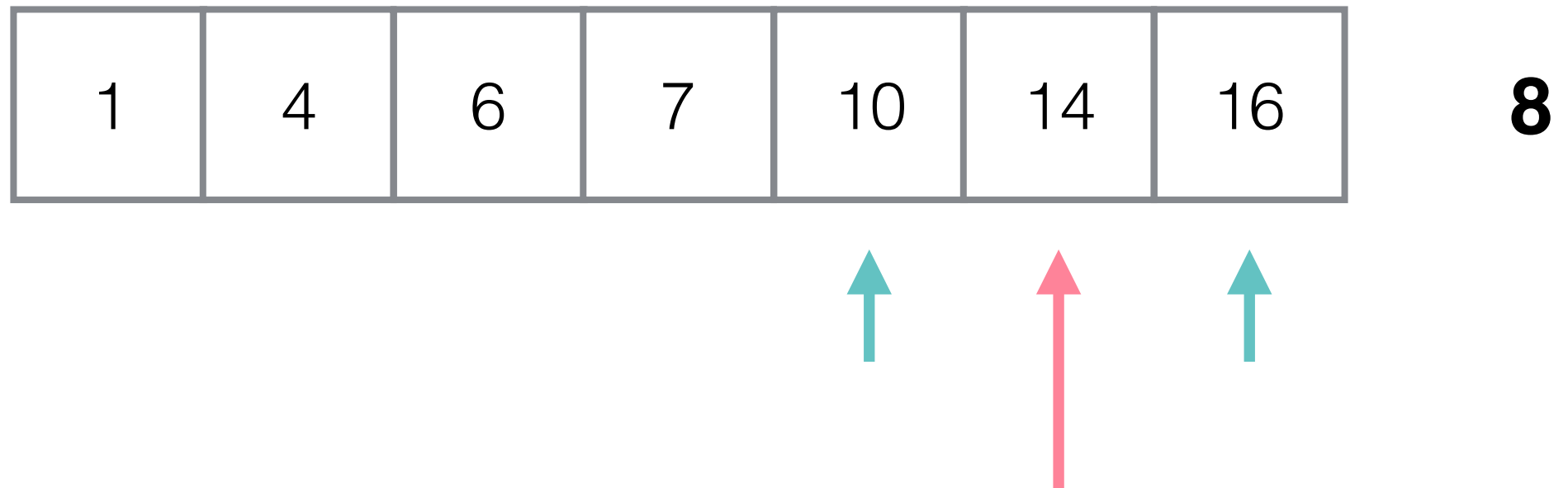
이진 탐색의 구현



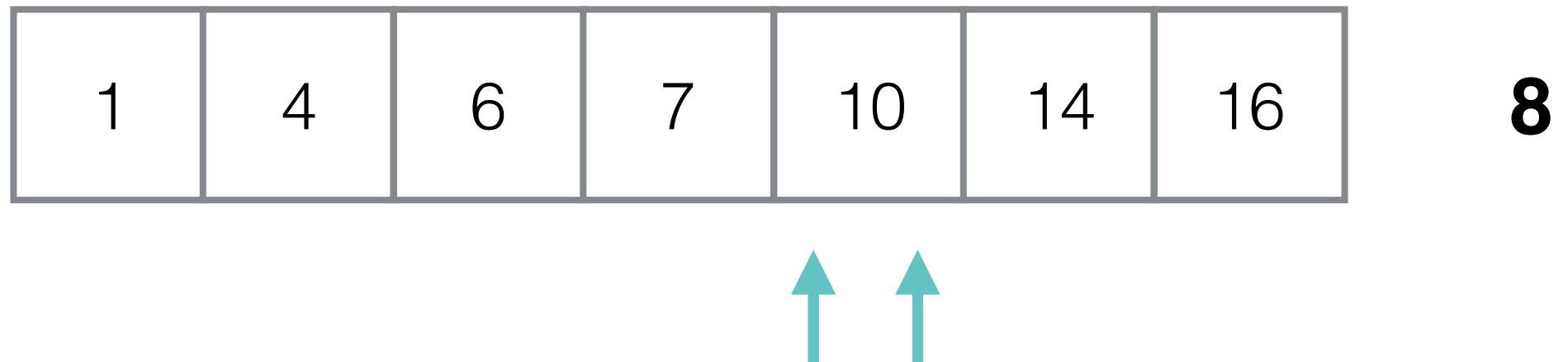
이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현

1	4	6	7	10	14	16
---	---	---	---	----	----	----

8



없음!

구현을 **한번에** 제대로 하려면

구현을 **한번에** 제대로 하려면

변수 및 함수의 **의미를 명확히** 한다

명확하지 않으면 200% 말리게 되어있음

구현을 **한번에** 제대로 하려면

변수 및 함수의 **의미를 명확히 한다**

명확하지 않으면 200% 말리게 되어있음

의미를 되새기면서 **천천히** 구현해야 한다

빠르게 구현해도 200% 말리게 되어있음

구현을 **한번에** 제대로 하려면

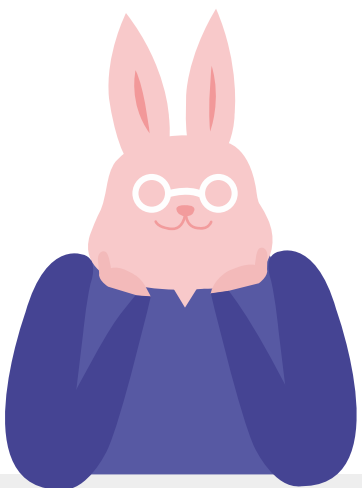
변수 및 함수의 **의미를 명확히 한다**

명확하지 않으면 200% 말리게 되어있음

의미를 되새기면서 **천천히** 구현해야 한다

빠르게 구현해도 200% 말리게 되어있음

강사의 코딩을 따라하면서 연습하자

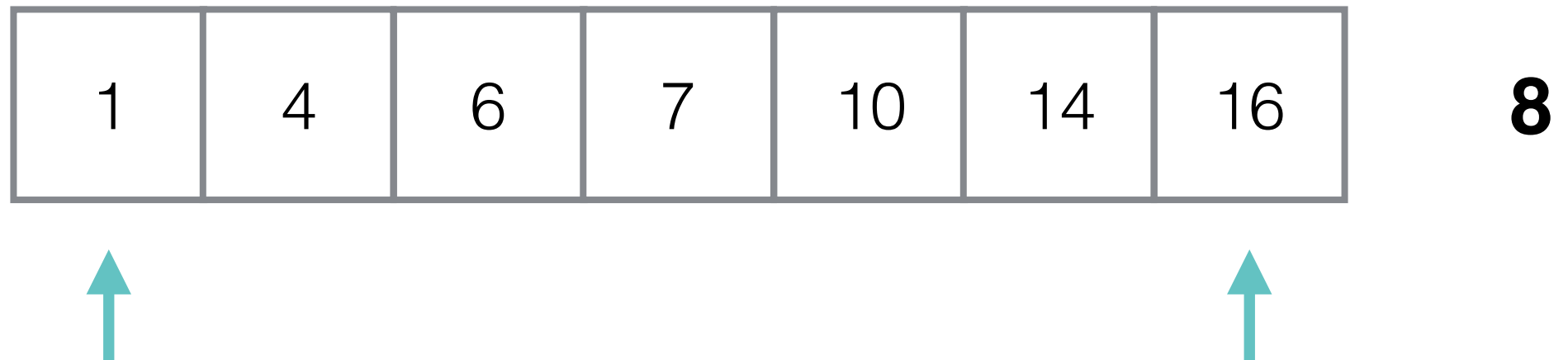


이진 탐색의 구현

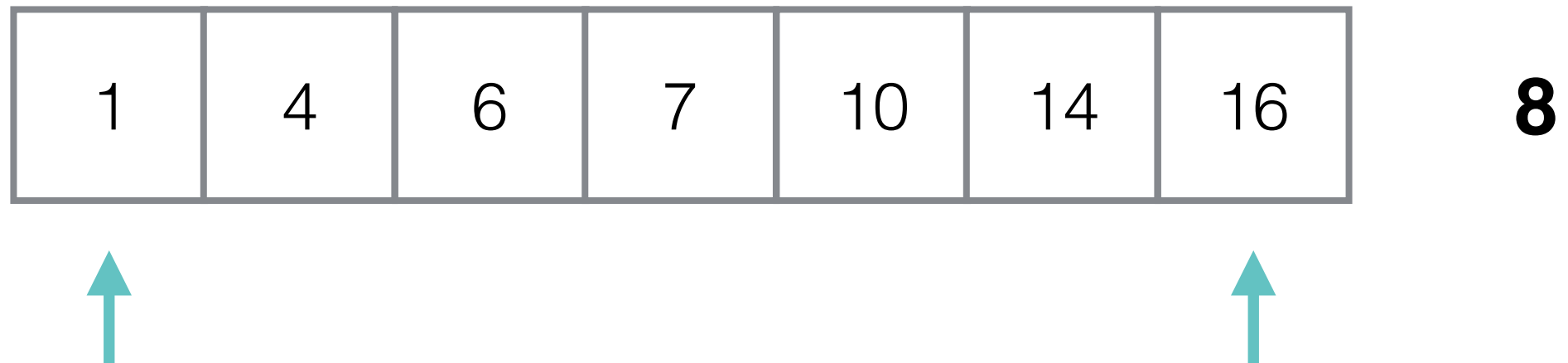
1	4	6	7	10	14	16
---	---	---	---	----	----	----

8

이진 탐색의 구현



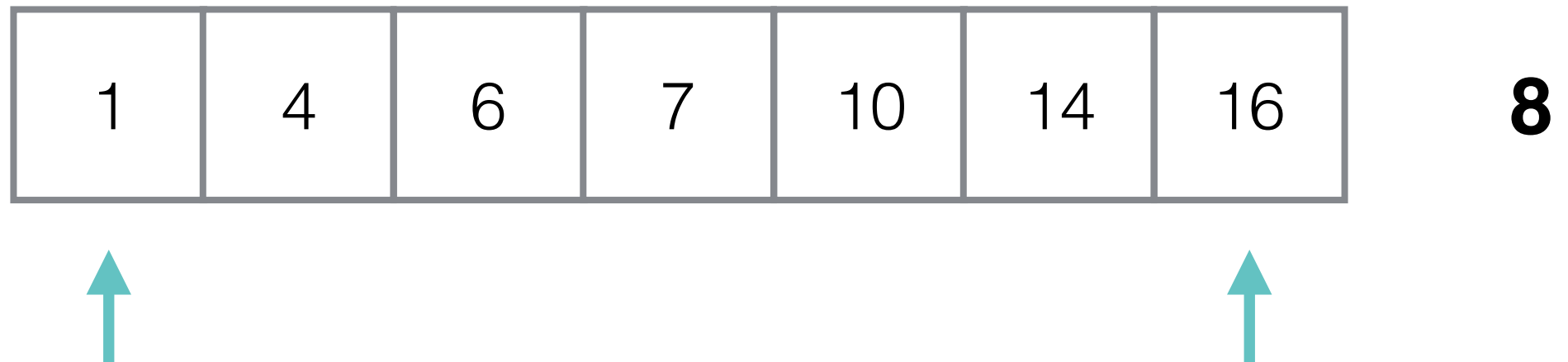
이진 탐색의 구현



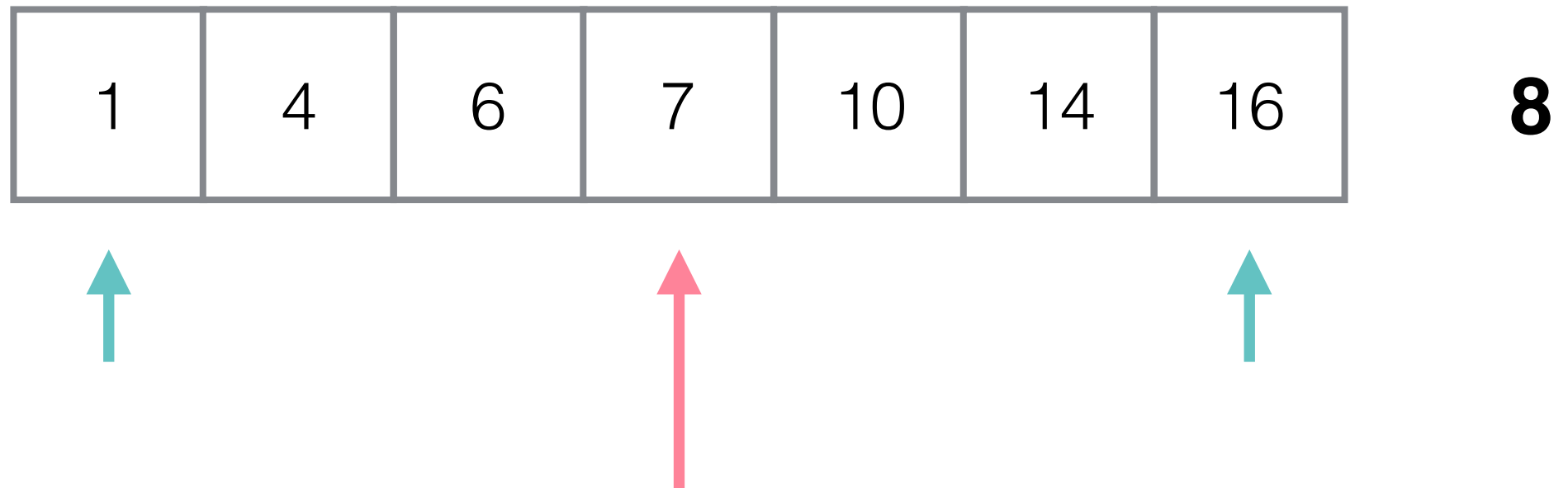
왼쪽 : 항상 작은 값을 가리킨다

오른쪽 : 항상 큰 값을 가리킨다

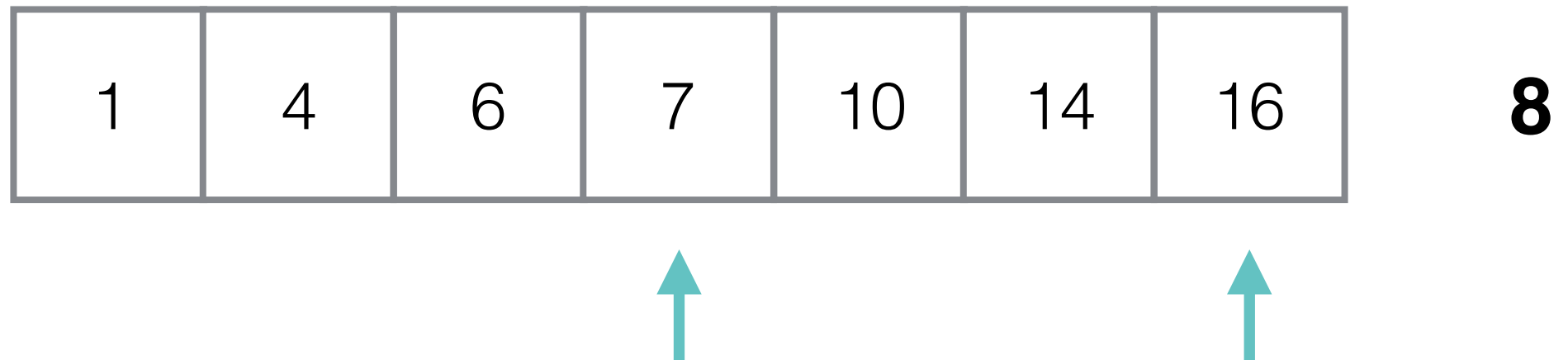
이진 탐색의 구현



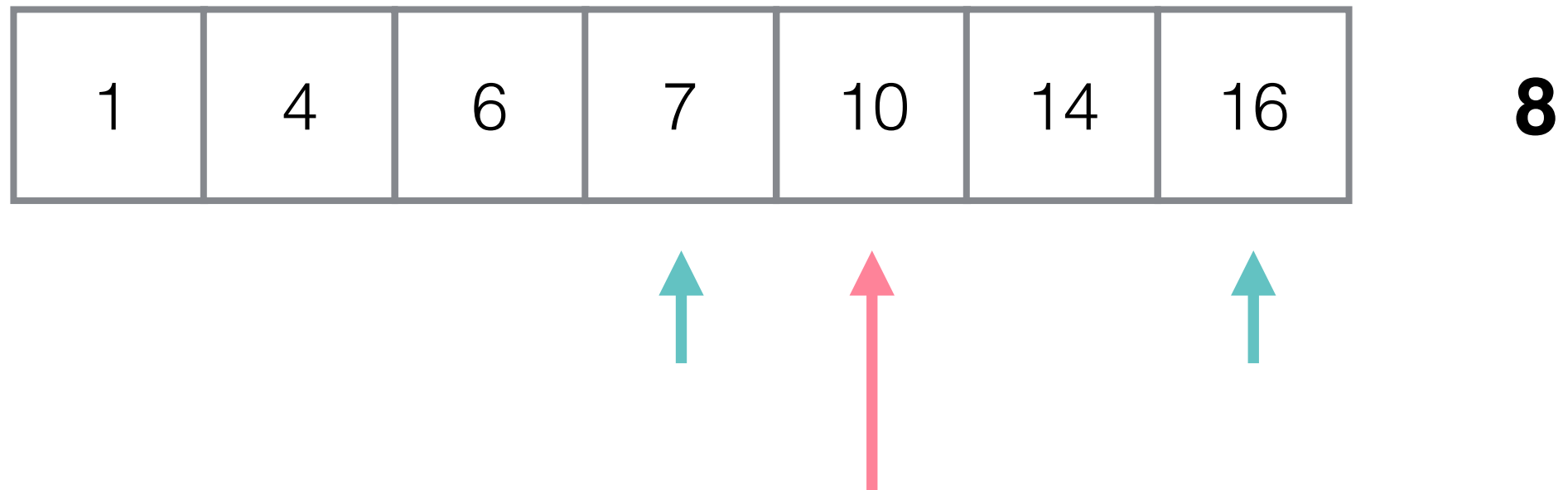
이진 탐색의 구현



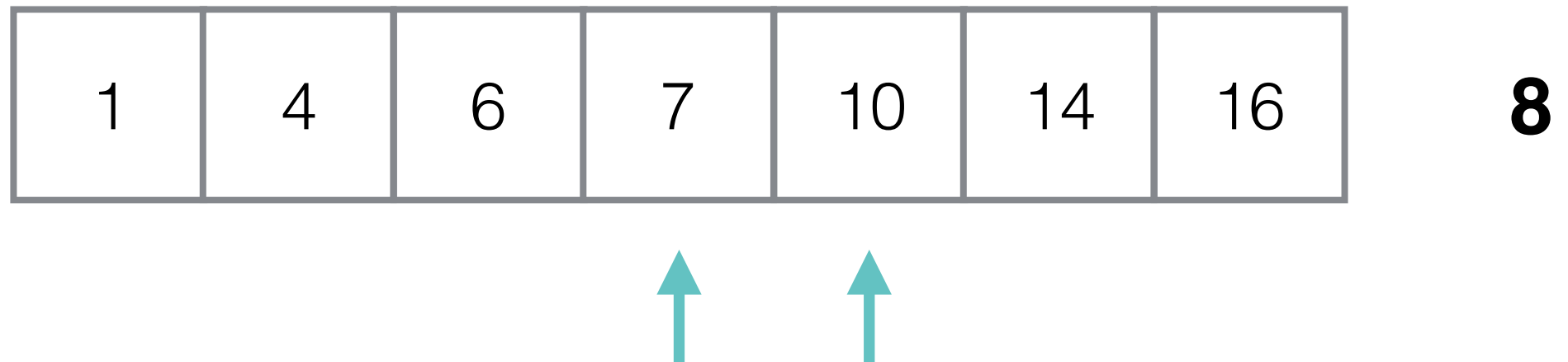
이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현



이진 탐색의 구현

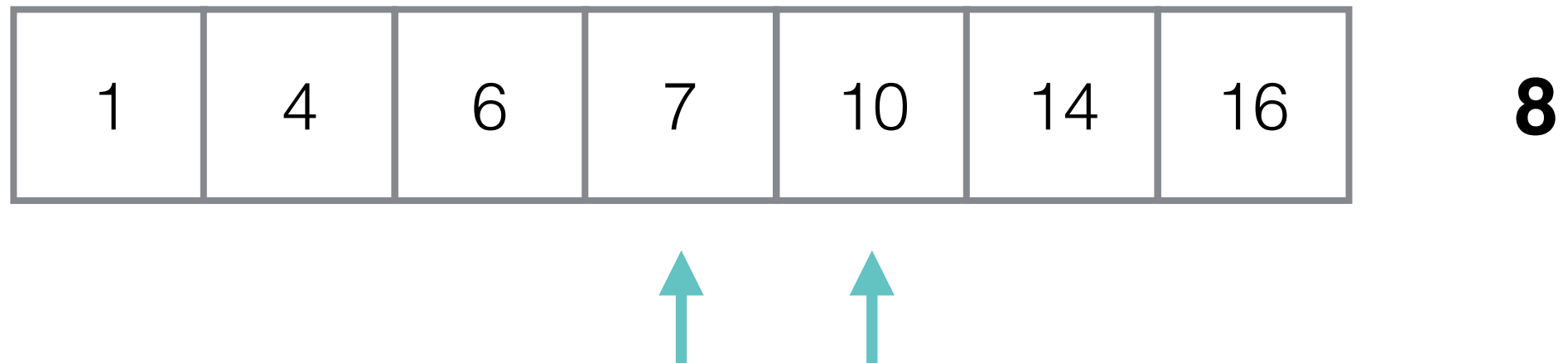
1	4	6	7	10	14	16
---	---	---	---	----	----	----

8



없음!

이진 탐색의 구현



없음!

왼쪽이 가리키는 값과 오른쪽이 가리키는 값에도 의미가 있음

재귀함수의 올바른 디자인 및 해석

재귀함수를 디자인하기 위해서는 다음 세 가지 단계를 명심하자

1. 함수의 정의를 명확히 한다.
2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.
3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

[예제 1] 가장 가까운 값 찾기

1. 함수의 정의를 명확히 한다.

[예제 1] 가장 가까운 값 찾기

1. 함수의 정의를 명확히 한다.

getNearest(data, m)

data에서

m보다 작거나 같은 값 중 최댓값,
m보다 큰 값 중 최솟값을 반환하는 함수

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 1$$



[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 1$$



$(-\infty, 16)$

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$

10	14	8
----	----	----------

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$

10	14
----	----

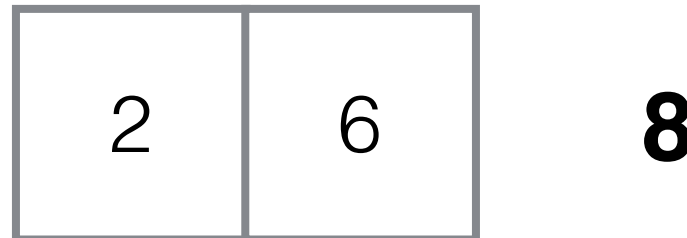
8

$(-\infty, 10)$

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$



[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$

2	6
---	---

8

(6, ∞)

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$

7	10	8
---	----	----------

[예제 1] 가장 가까운 값 찾기

2. 기저 조건(Base condition)에서 함수가 제대로 동작하게 작성한다.

$$n = 2$$

7	10
---	----

8

(7, 10)

[예제 1] 가장 가까운 값 찾기

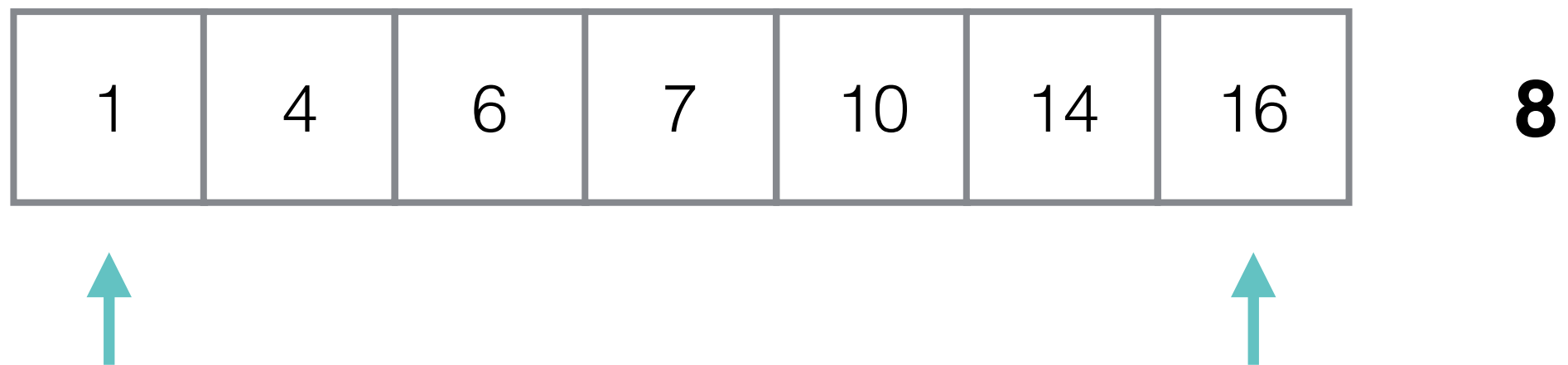
3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

1	4	6	7	10	14	16
---	---	---	---	----	----	----

8

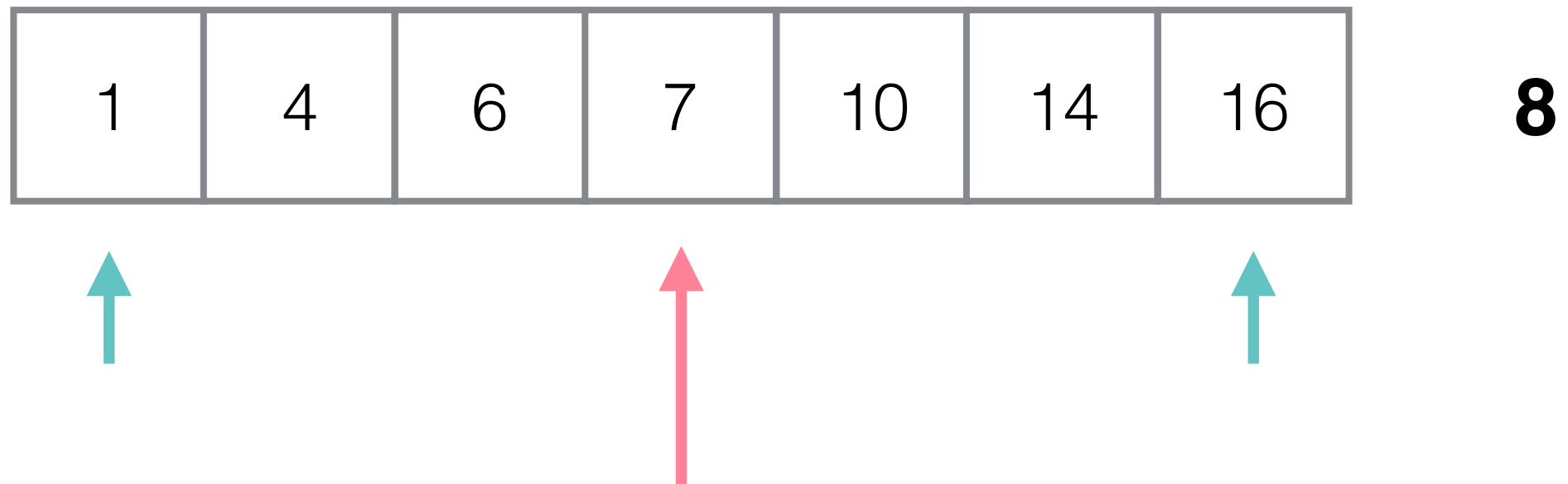
[예제 1] 가장 가까운 값 찾기

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.



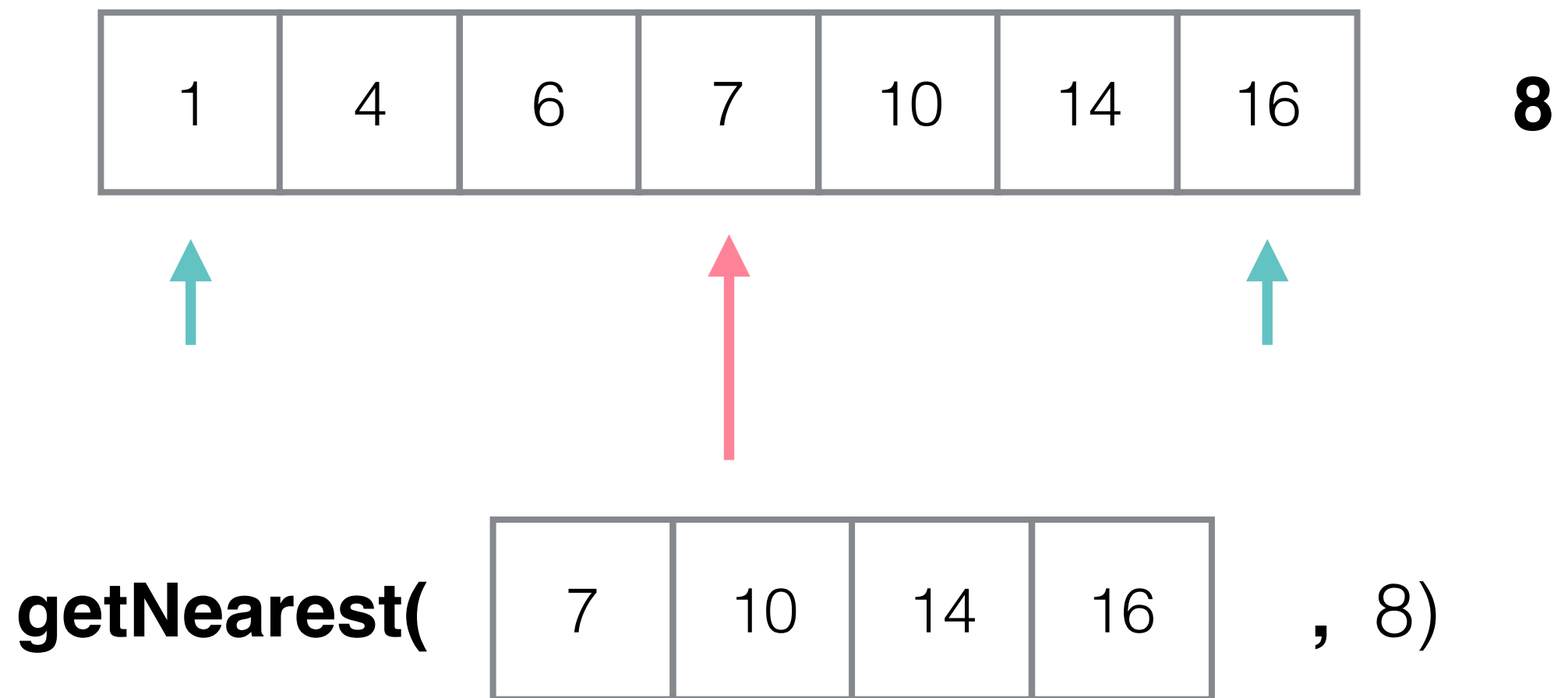
[예제 1] 가장 가까운 값 찾기

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.

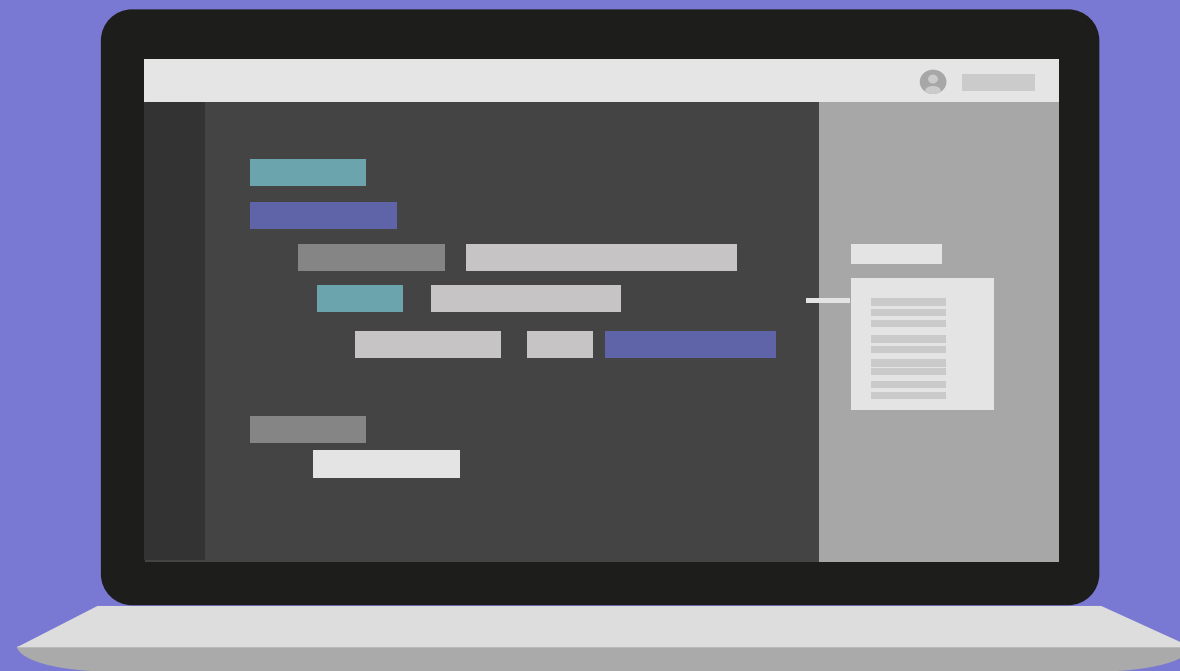


[예제 1] 가장 가까운 값 찾기

3. 그 후, 함수가 (작은 input에 대하여) 제대로 동작한다고 가정하고 함수를 완성한다.



[예제 1] 가장 가까운 값 찾기



`/* elice */`

[예제 2] 거듭제곱 구하기

정렬된 n 개의 숫자 중 정수 m 과 가장 가까운 값을 찾아라
단, $1 \leq n \leq 100,000$

입력의 예

```
1 4 6 7 10 14 16
8
```

출력의 예

```
7
```

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$\text{getPower}(m, n) = m \times \text{getPower}(m, n-1)$$

$$\text{getPower}(m, 0) = 1$$

[예제 2] 거듭제곱 구하기

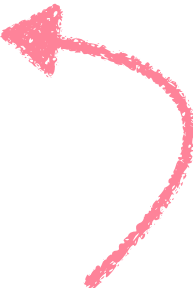
$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수



$$\text{getPower}(m, n) = m \times \text{getPower}(m, n-1)$$

$$\text{getPower}(m, 0) = 1$$



점화식

기저조건

(Base condition)

/* elice */

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    else :  
        return m * getPower(m, n-1)
```

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    else :  
        return m * getPower(m, n-1)
```

O(n)

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \quad n \text{이 짝수일 경우}$$

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$m^n = (m^{(n/2)})^2 \quad n \text{이 짝수일 경우}$$

$$(m^{n-1}) \times m \quad n \text{이 홀수일 경우}$$

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

getPower(m, n) =

n이 짝수

getPower(m, n) =

n이 홀수

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

$$\text{getPower}(m, n) = (\text{getPower}(m, n//2))^2 \quad n0 \text{이 짝수}$$

$$\text{getPower}(m, n) = m \times \text{getPower}(m, n-1) \quad n0 \text{이 홀수}$$

[예제 2] 거듭제곱 구하기

$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    elif n % 2 == 0 :  
        temp = getPower(m, n//2)  
        return temp * temp  
    else :  
        return getPower(m, n-1) * m
```

[예제 2] 거듭제곱 구하기

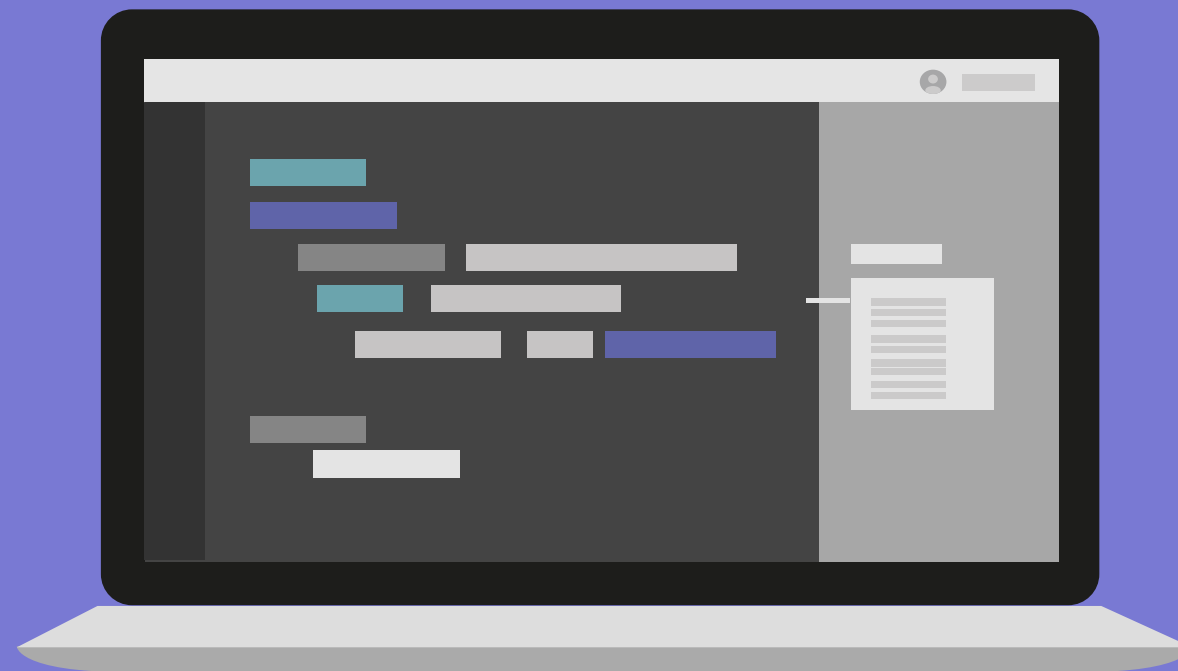
$$m^n = m \times m \times \dots \times m$$

getPower(m, n) : m^n 을 반환하는 함수

```
def getPower(m, n) :  
    if n == 0 :  
        return 1  
    elif n % 2 == 0 :  
        temp = getPower(m, n//2)  
        return temp * temp  
    else :  
        return getPower(m, n-1) * m
```

$O(\log n)$

[예제 2] 거듭제곱 구하기



```
/* elice */
```


03 분할정복법 (Divide & Conquer)

분할정복법 (Divide & Conquer)

분할정복법 (Divide & Conquer)

문제를 소문제로 분할

분할정복법 (Divide & Conquer)

문제를 소문제로 분할

각각의 소문제를 해결

분할정복법 (Divide & Conquer)

문제를 소문제로 분할

각각의 소문제를 해결

소문제의 해결 결과를 이용하여 전체 문제를 해결

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	7	4	2	10	19	2	4	5	3	1	5
---	---	---	---	----	----	---	---	---	---	---	---

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	7	4	2	10	19	2	4	5	3	1	5
---	---	---	---	----	----	---	---	---	---	---	---

pivot

퀵정렬 (Quick Sort)

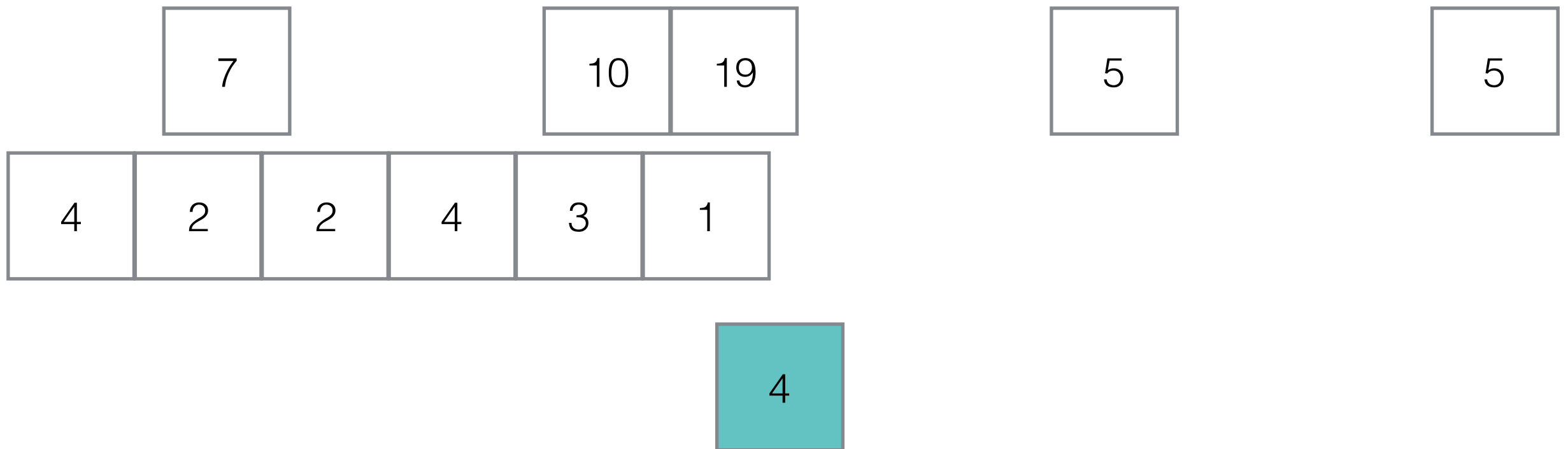
재귀호출을 이용한 대표적인 정렬

7	4	2	10	19	2	4	5	3	1	5
---	---	---	----	----	---	---	---	---	---	---



퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	2	2	4	3	1
---	---	---	---	---	---

7	10	19	5	5
---	----	----	---	---

4

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

4	2	2	4	3	1	4	7	10	19	5	5
---	---	---	---	---	---	---	---	----	----	---	---

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

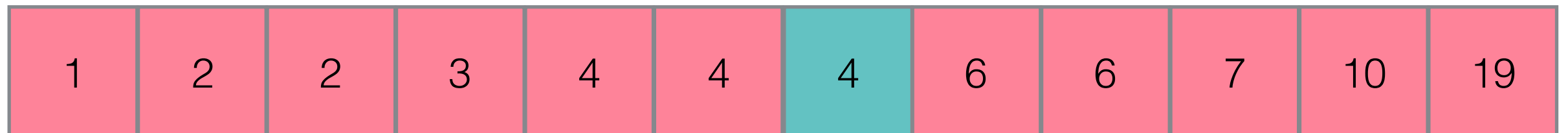
4	2	2	4	3	1	4	7	10	19	5	5
---	---	---	---	---	---	---	---	----	----	---	---

Quicksort!

Quicksort!

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



Quicksort!

Quicksort!

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

1	2	2	3	4	4	4	6	6	7	10	19
---	---	---	---	---	---	---	---	---	---	----	----

합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

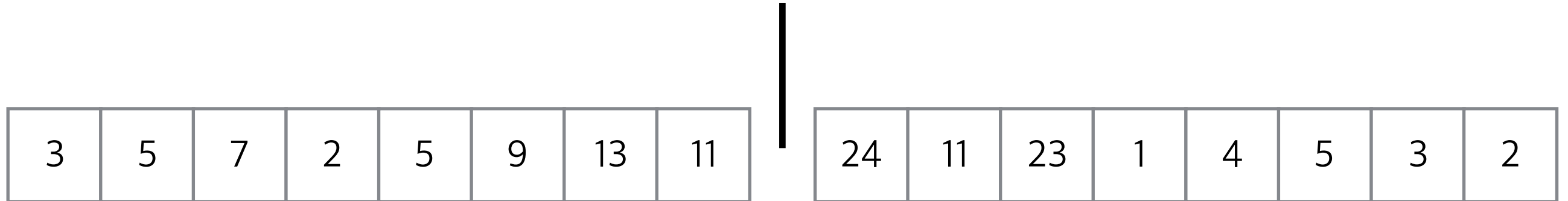
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

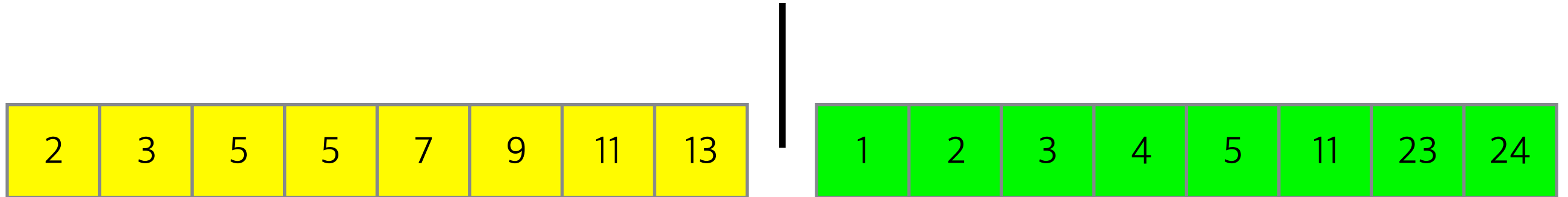
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



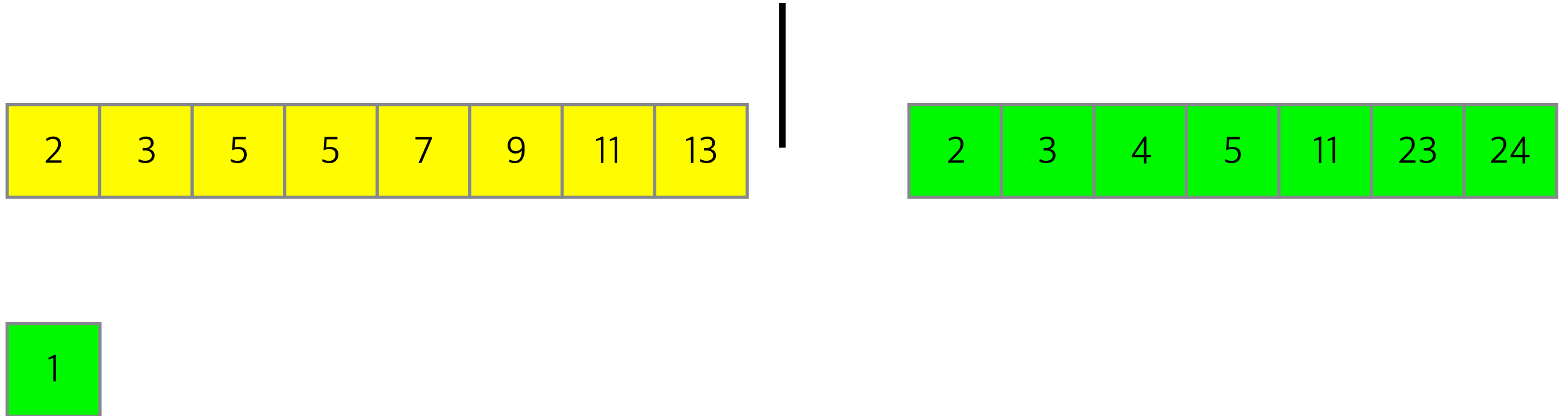
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



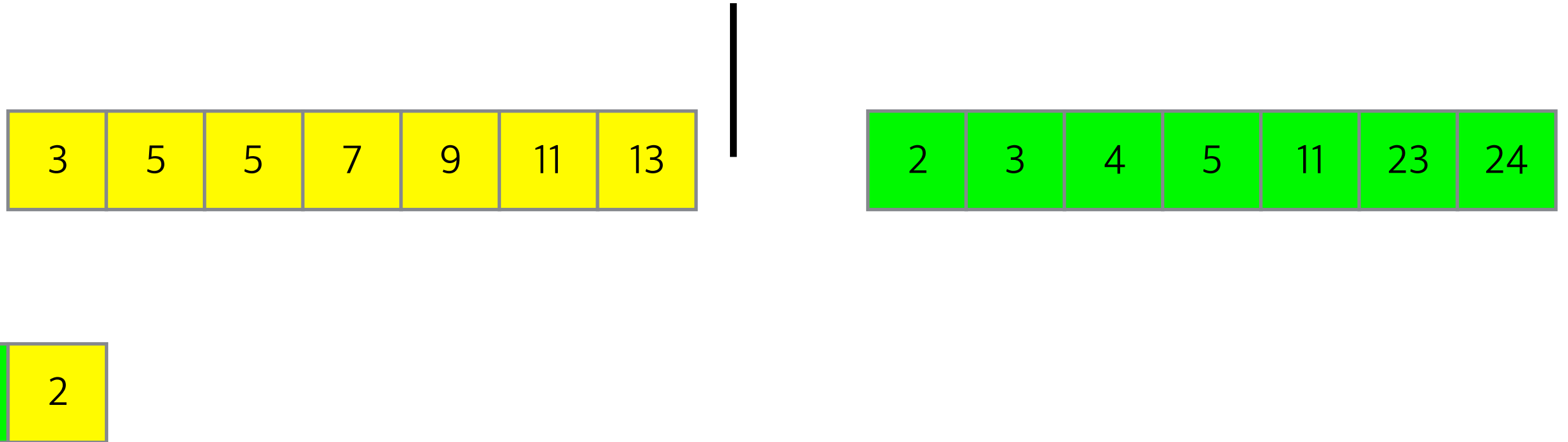
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



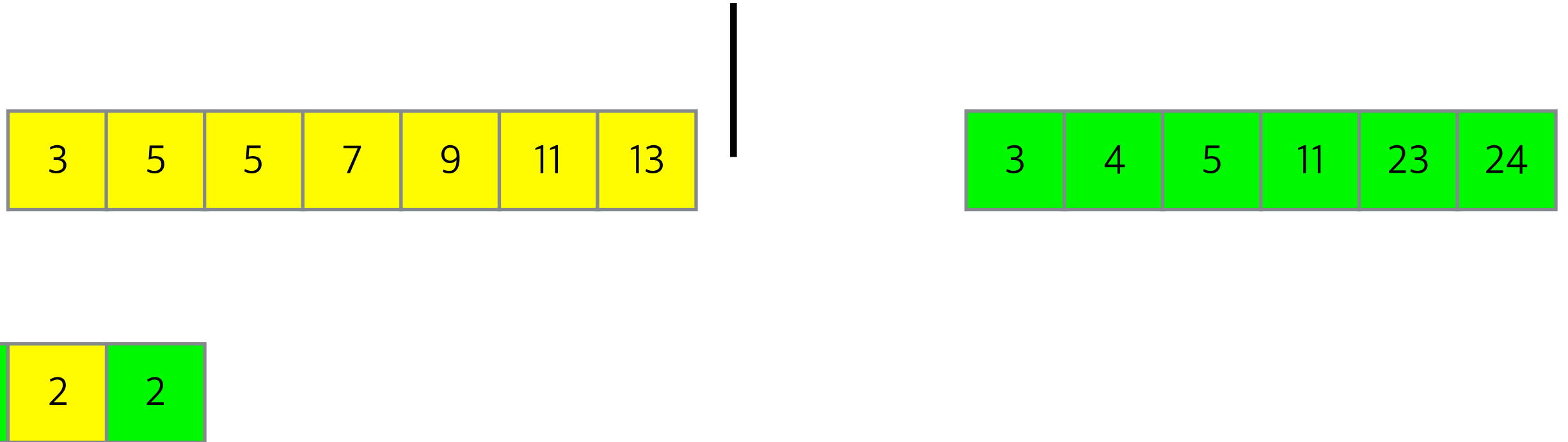
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



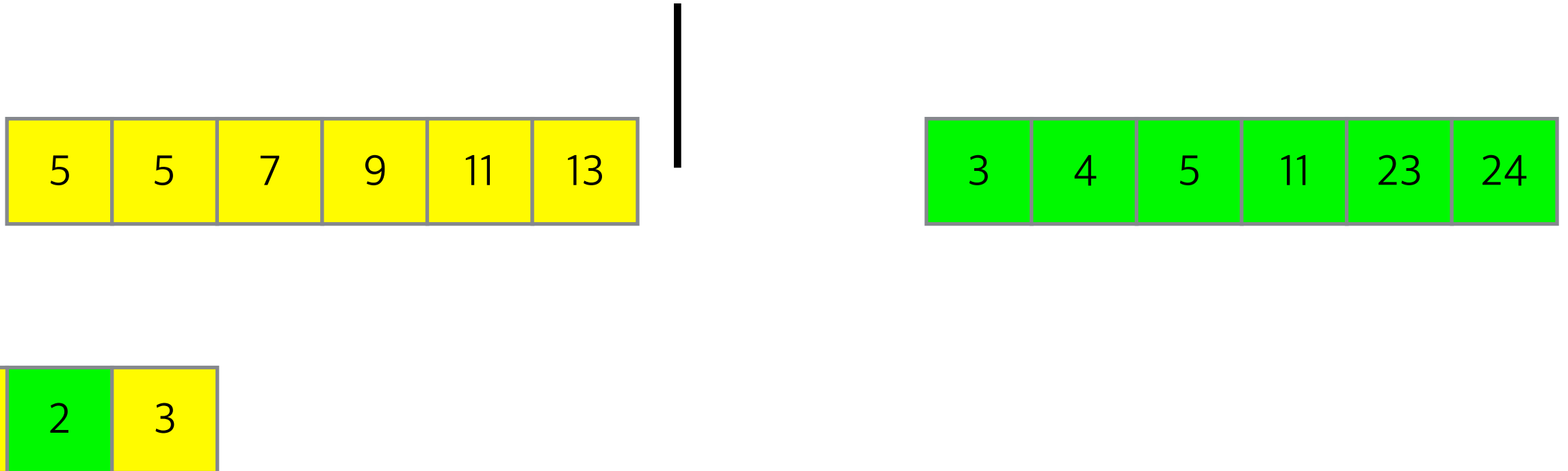
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



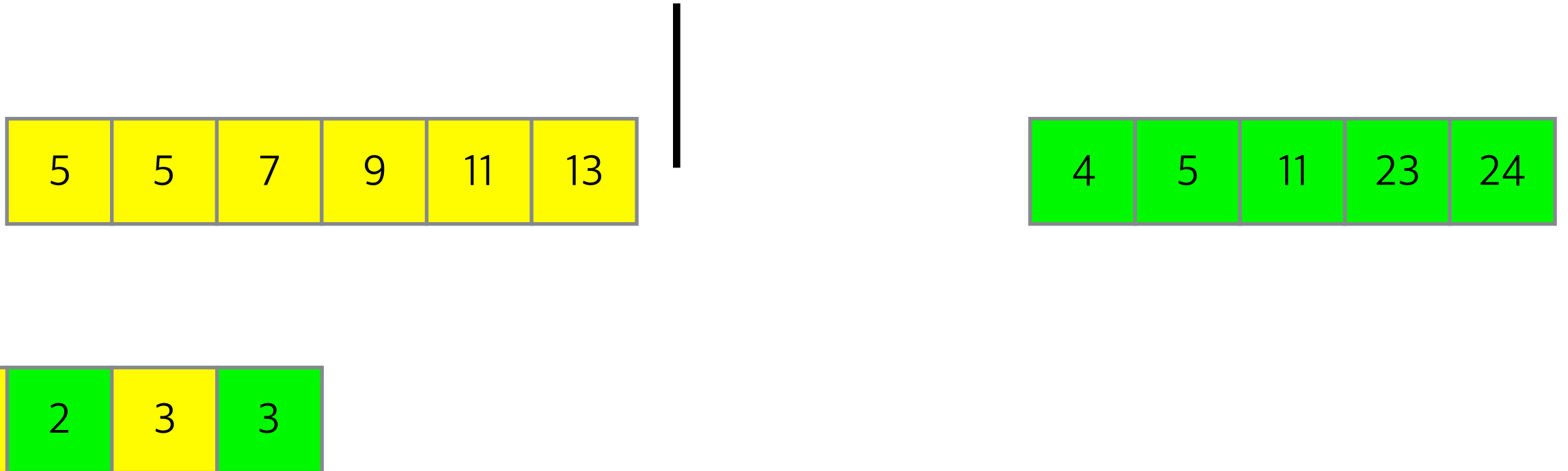
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



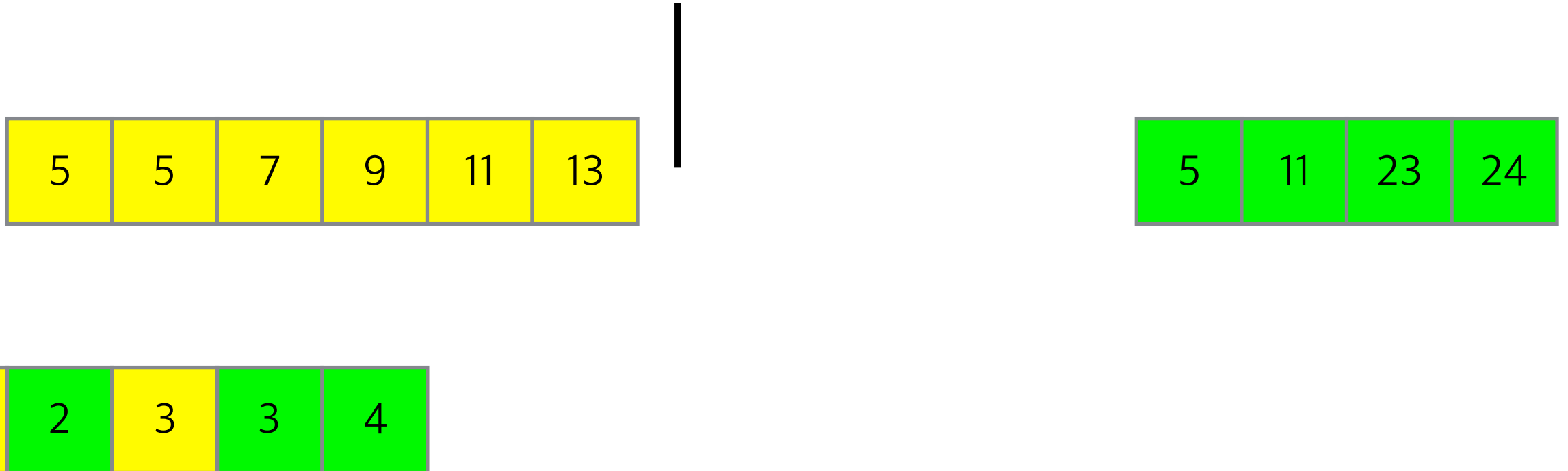
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



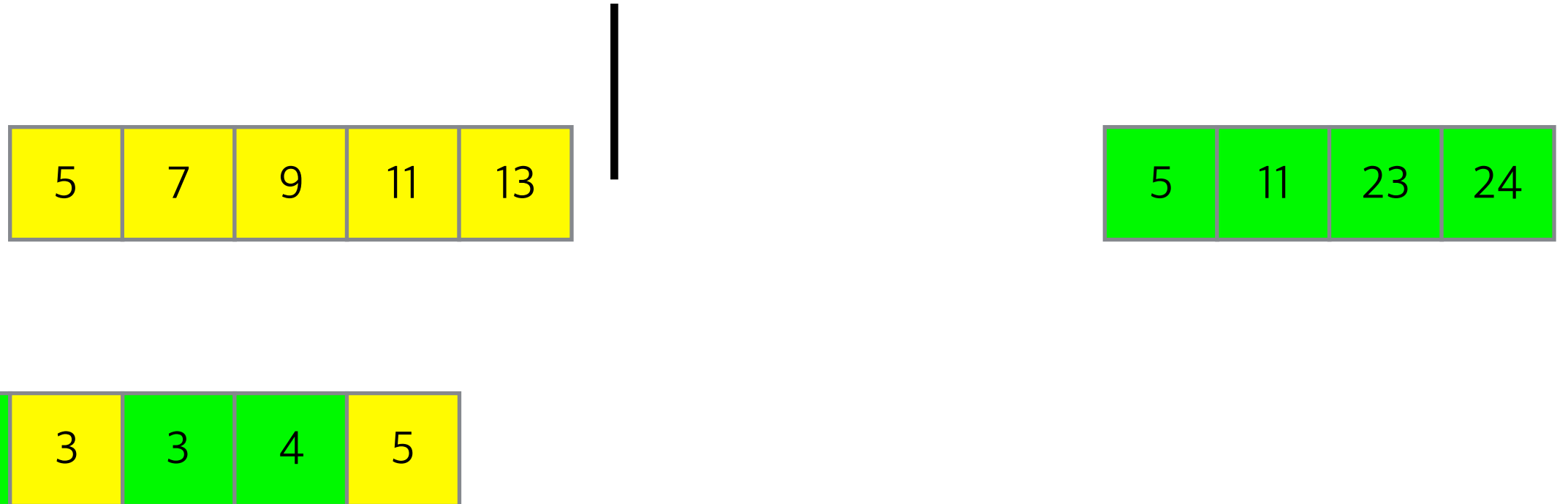
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



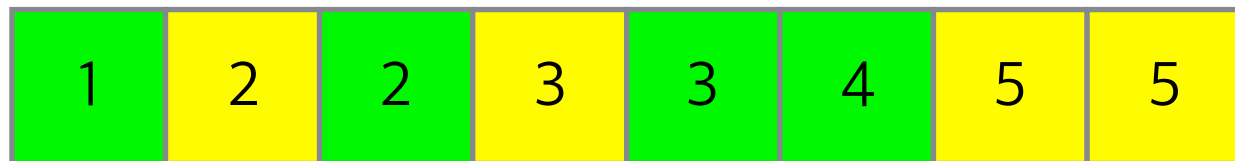
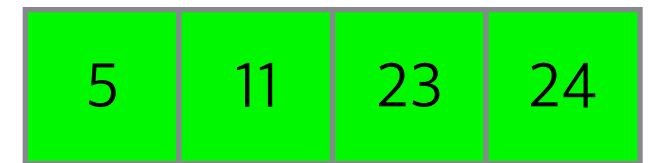
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



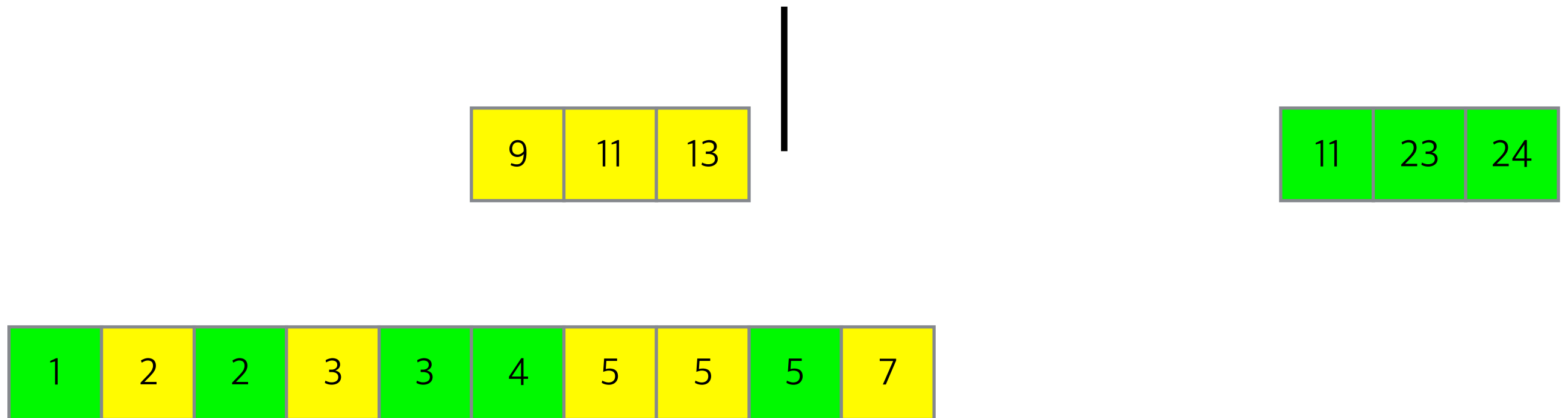
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



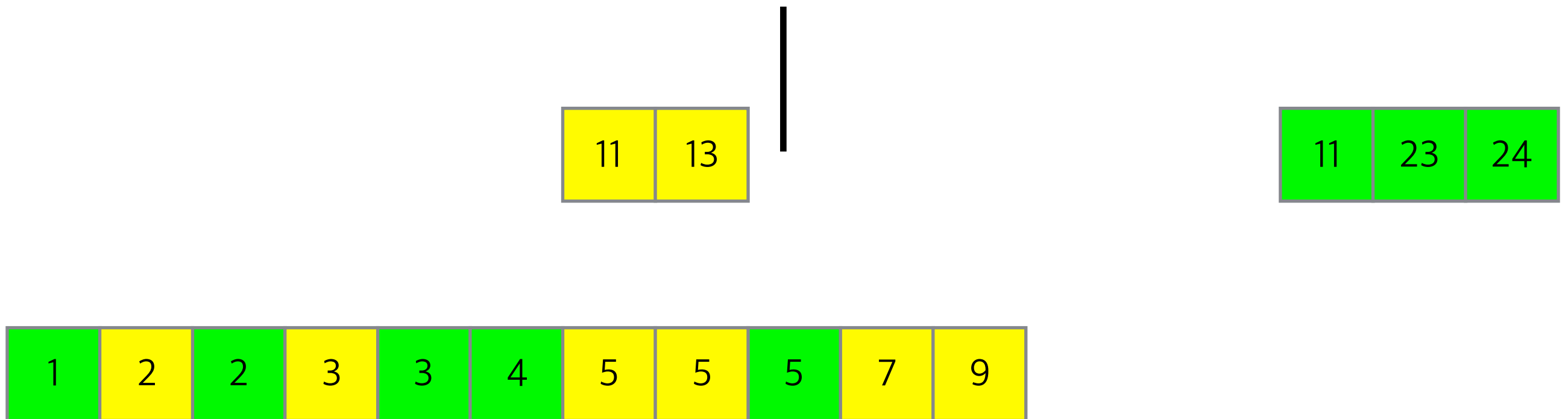
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



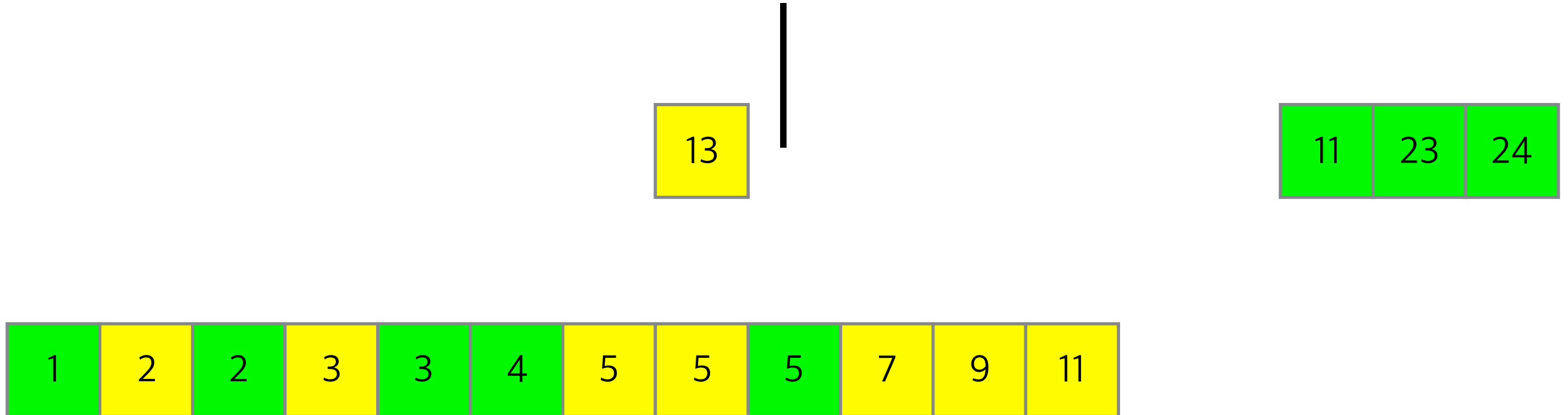
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



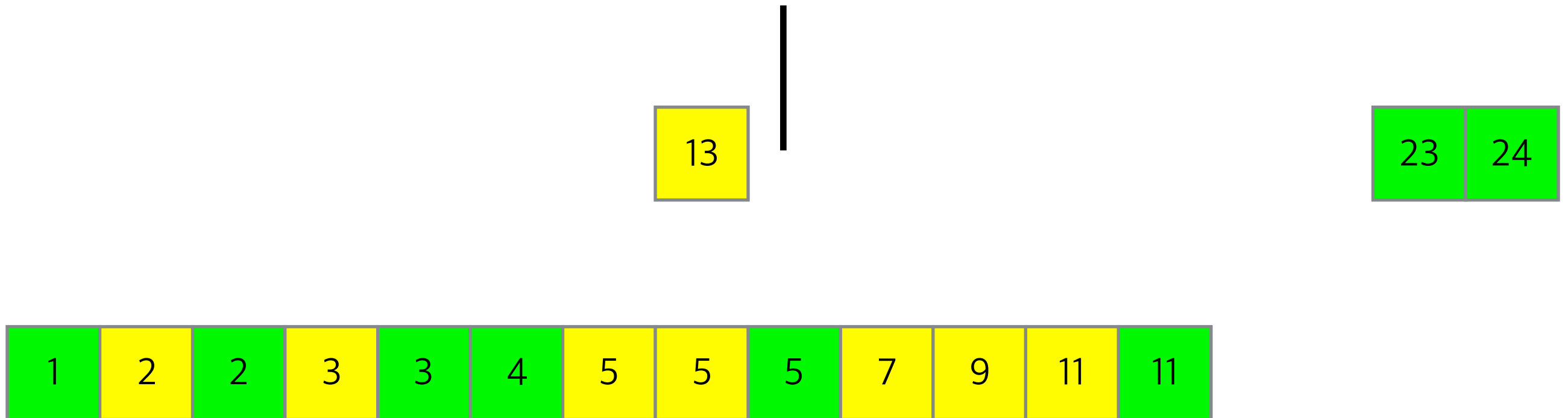
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

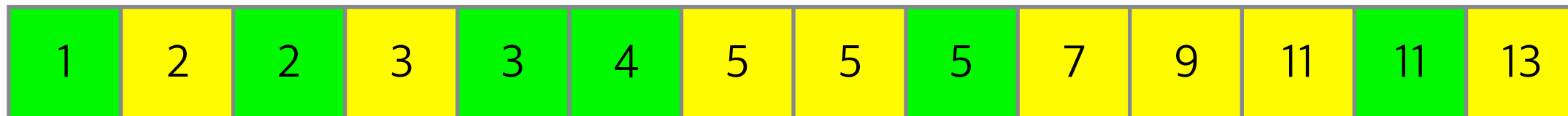


합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

|

23	24
----	----



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

|

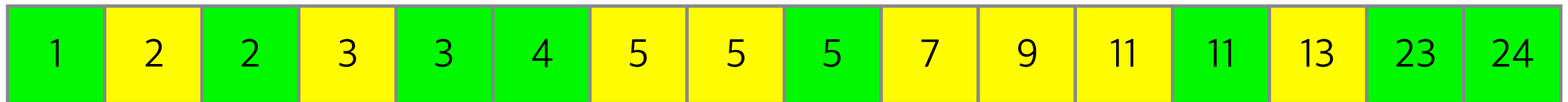
24



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

|



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

1	2	2	3	3	4	5	5	5	7	9	11	11	13	23	24
---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

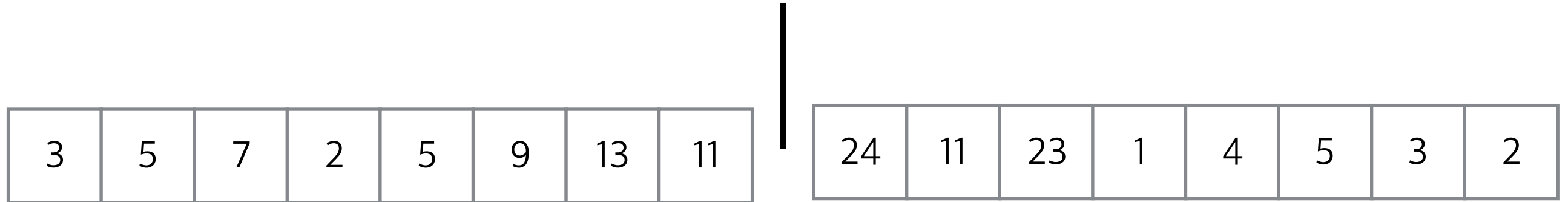
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

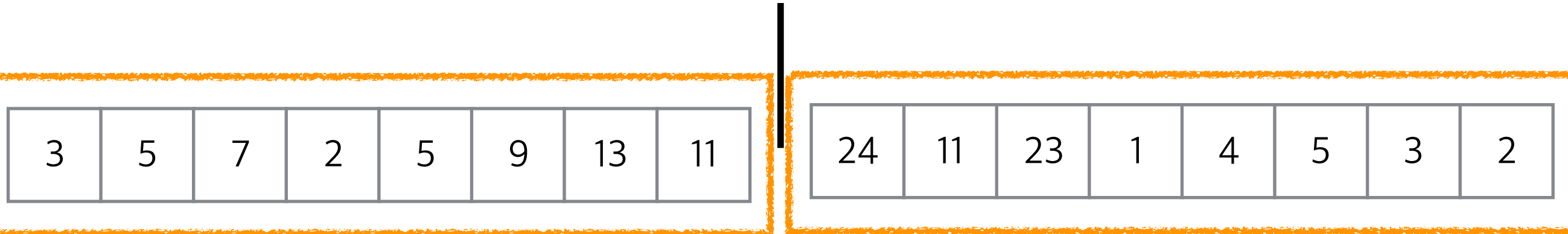
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



합병정렬 (Merge Sort)

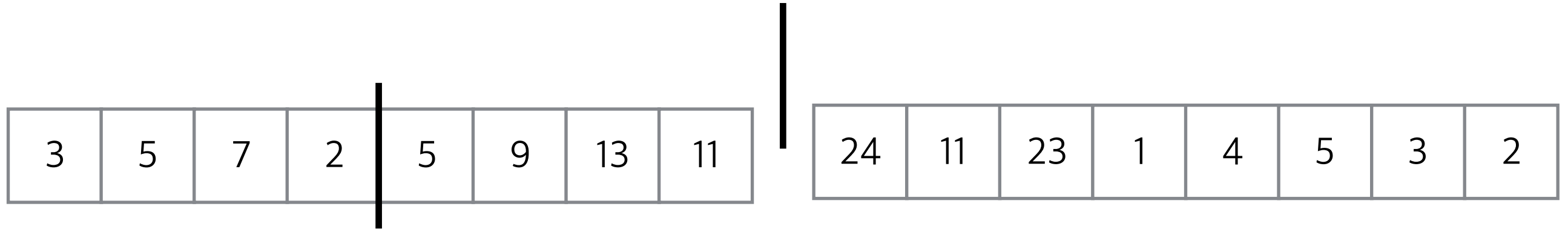
재귀호출을 이용한 대표적인 정렬



각각을 Merge sort로 정렬!

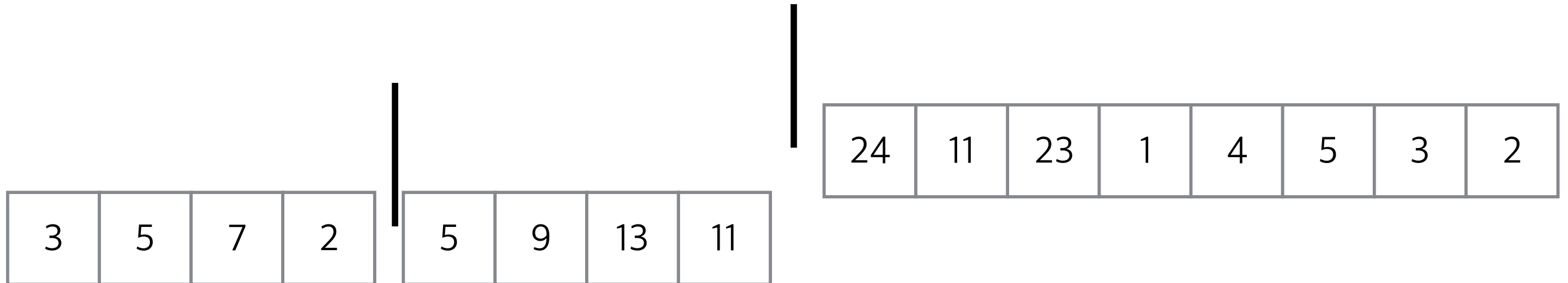
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



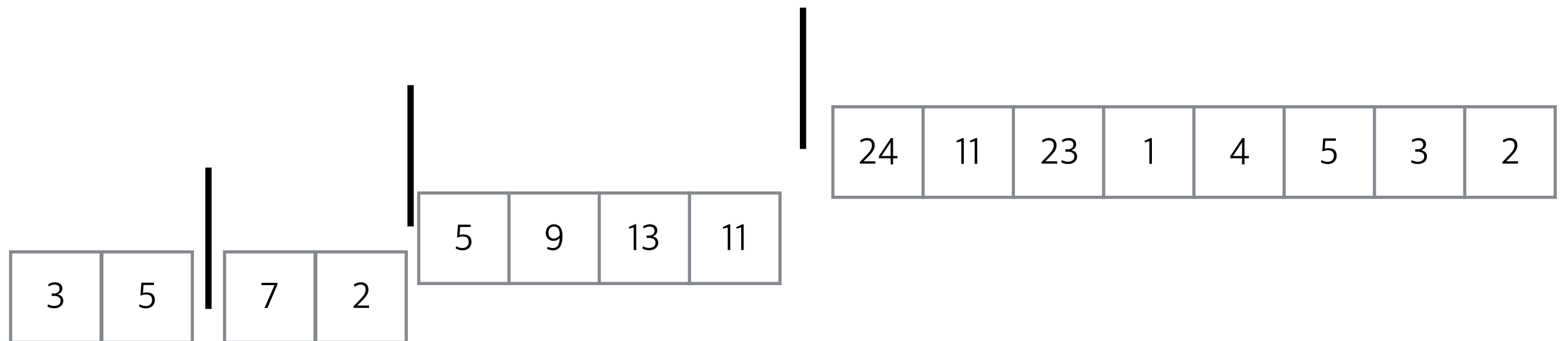
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



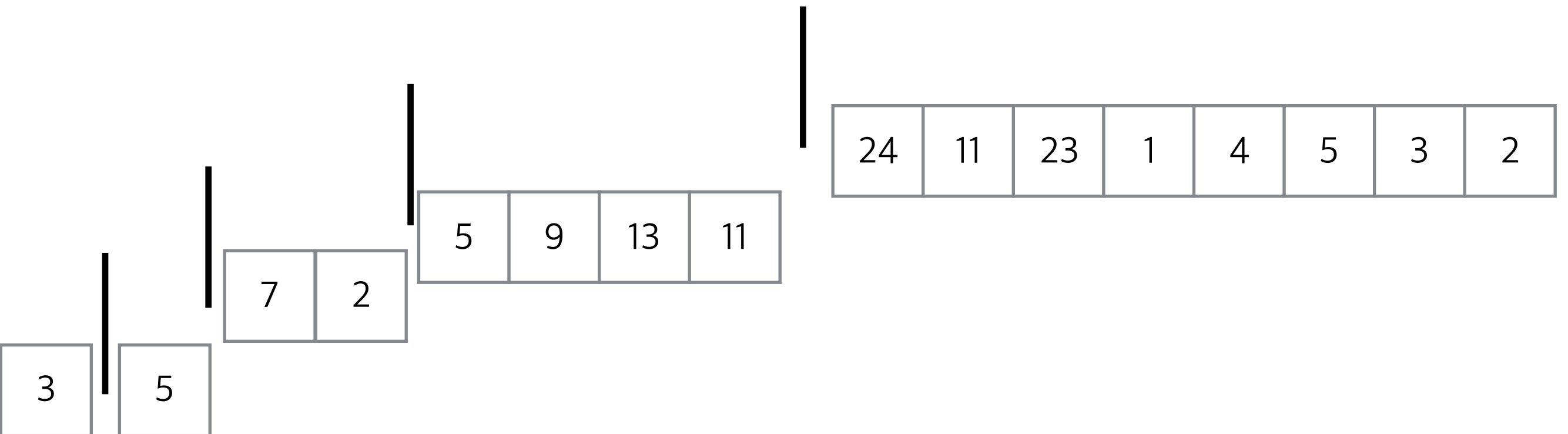
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



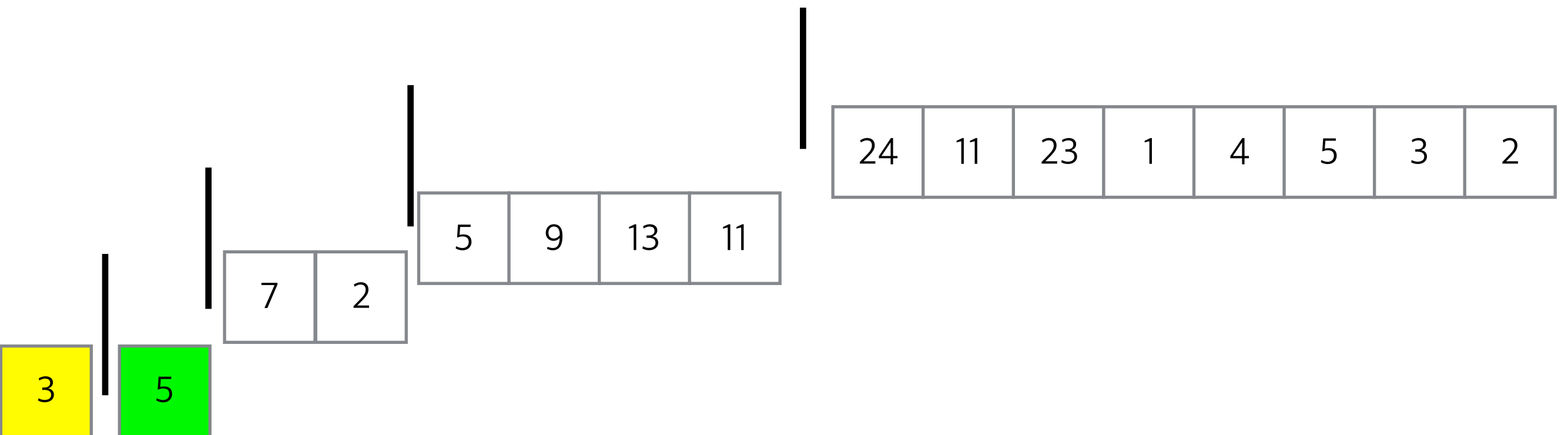
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



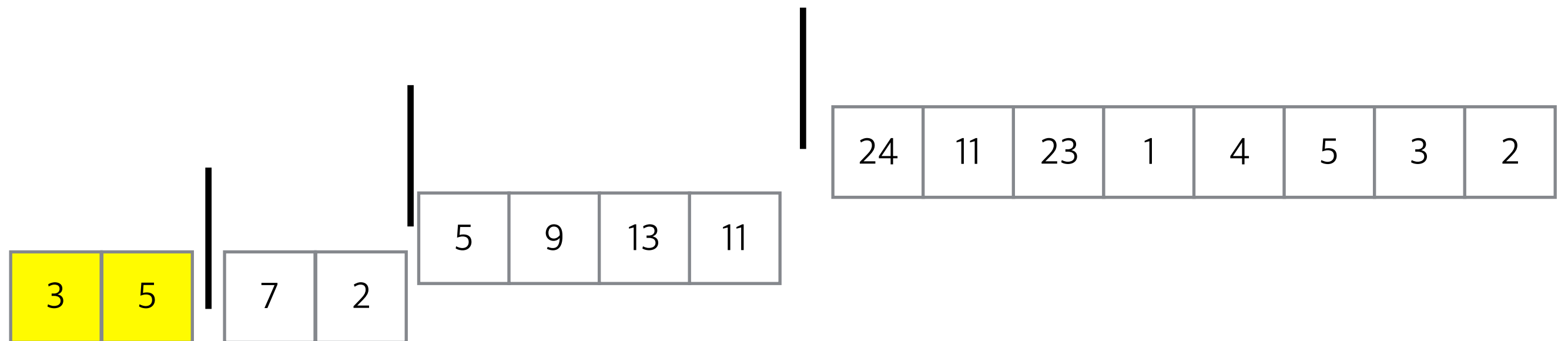
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



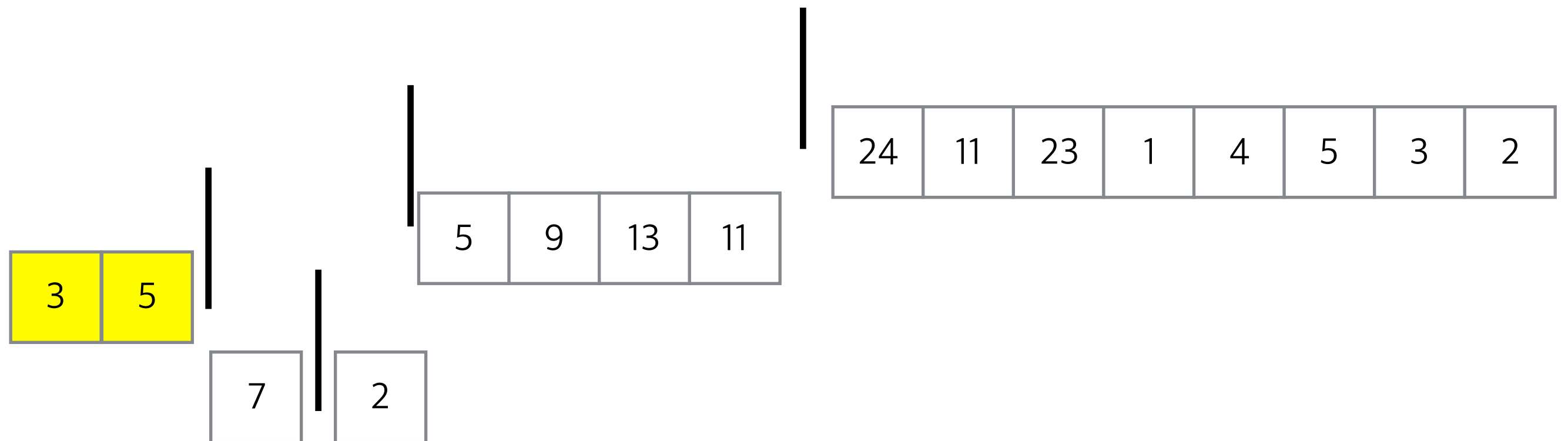
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



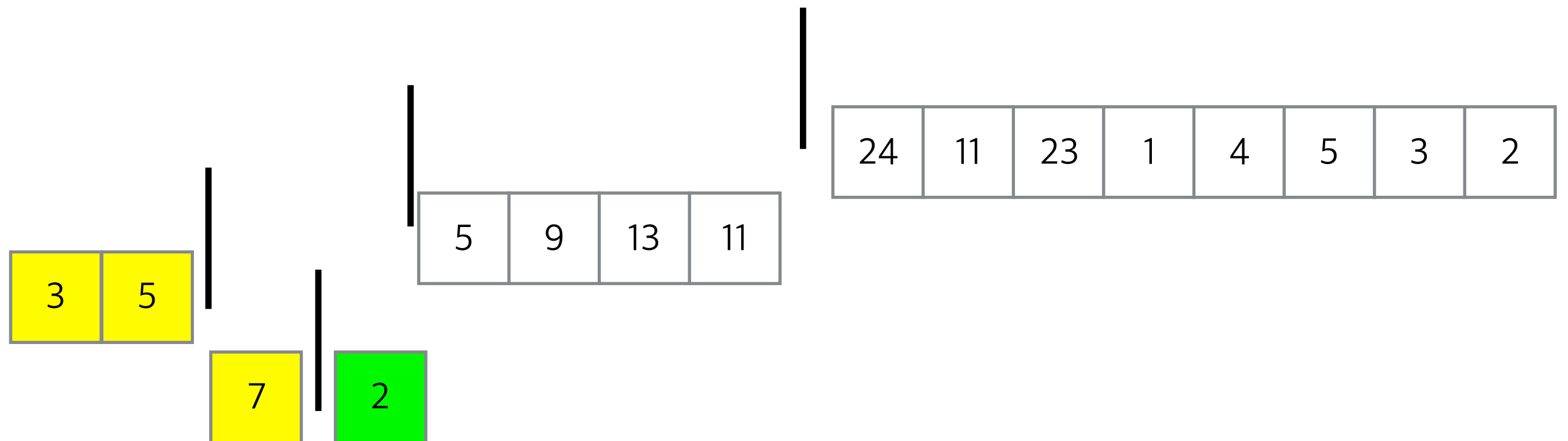
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



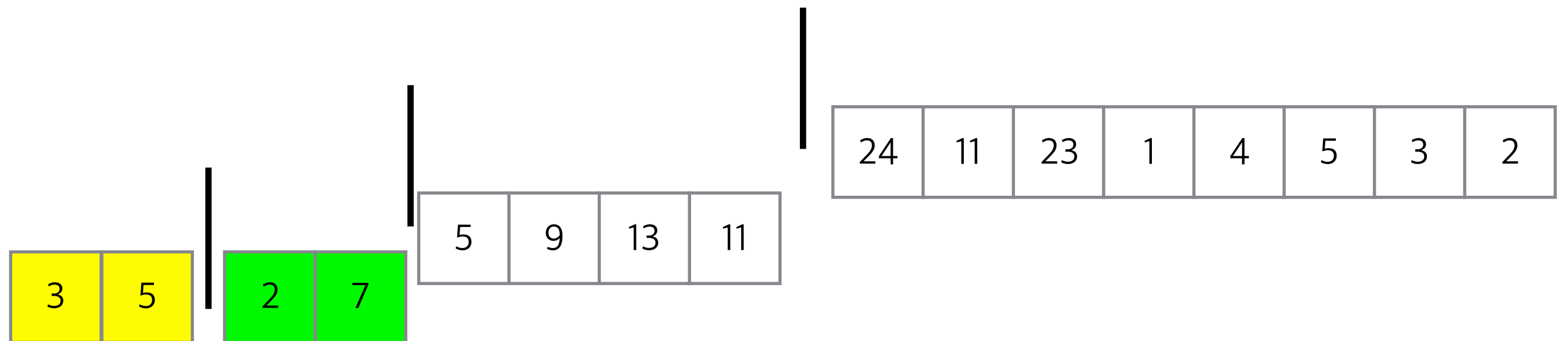
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



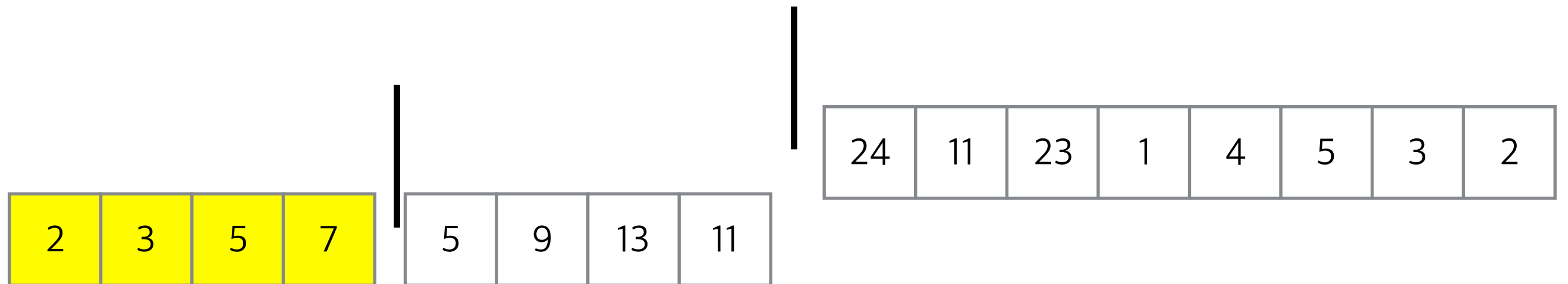
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



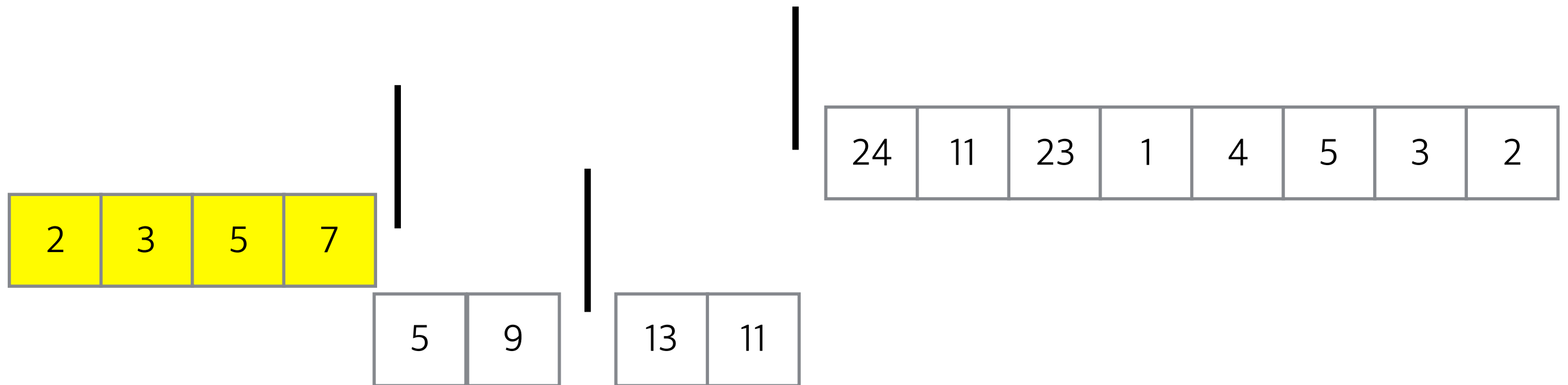
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



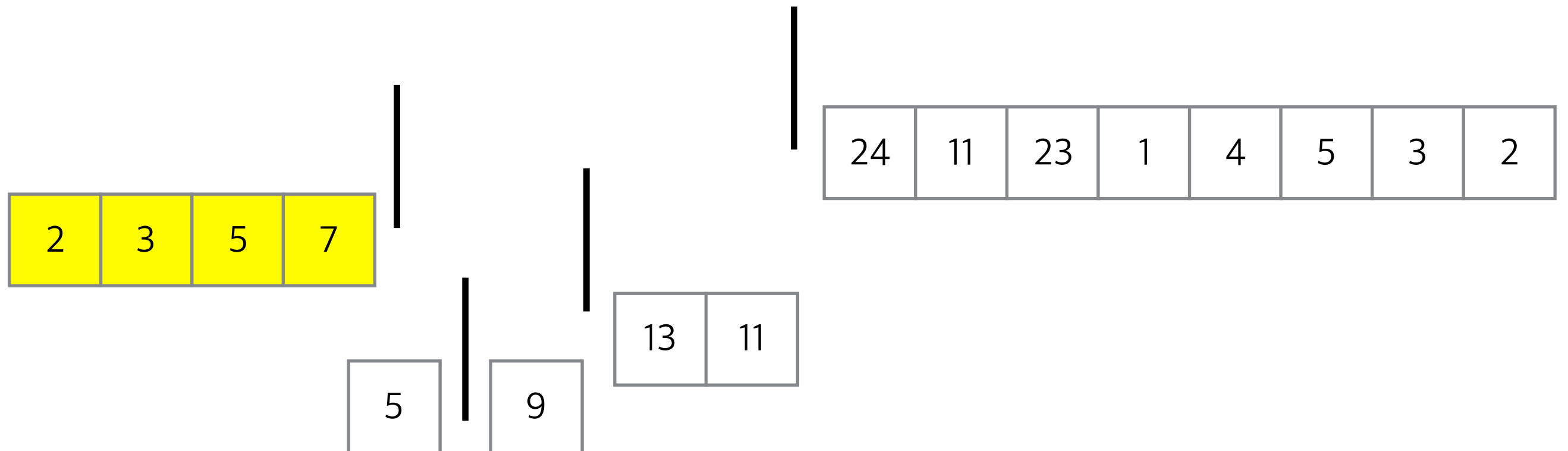
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



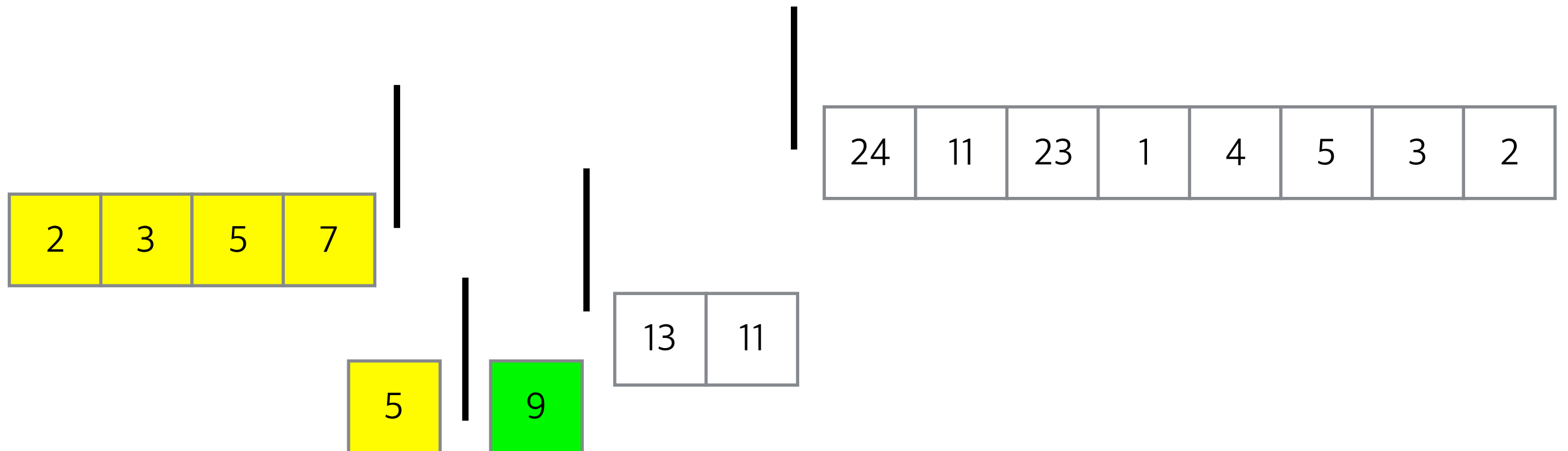
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



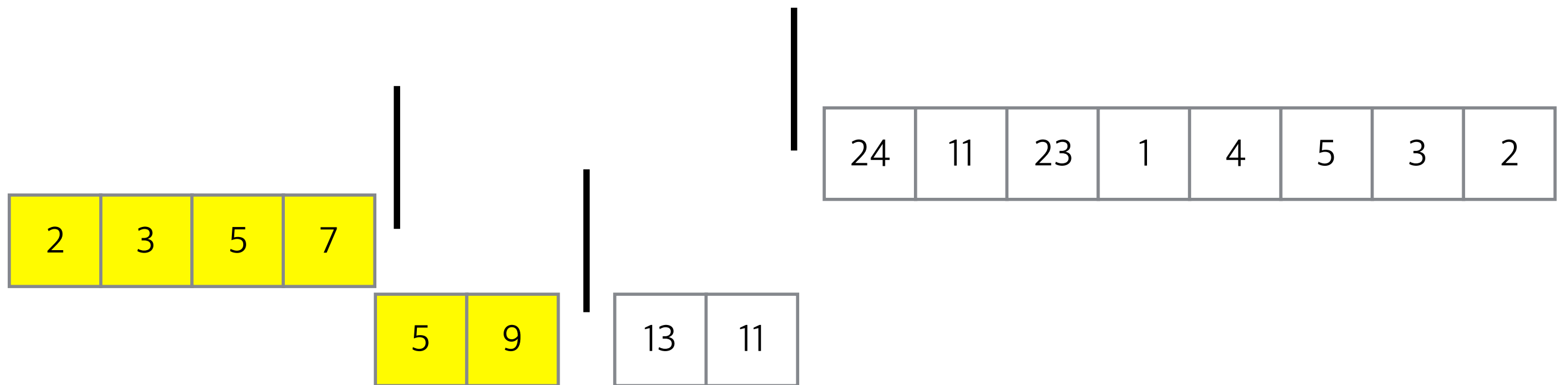
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



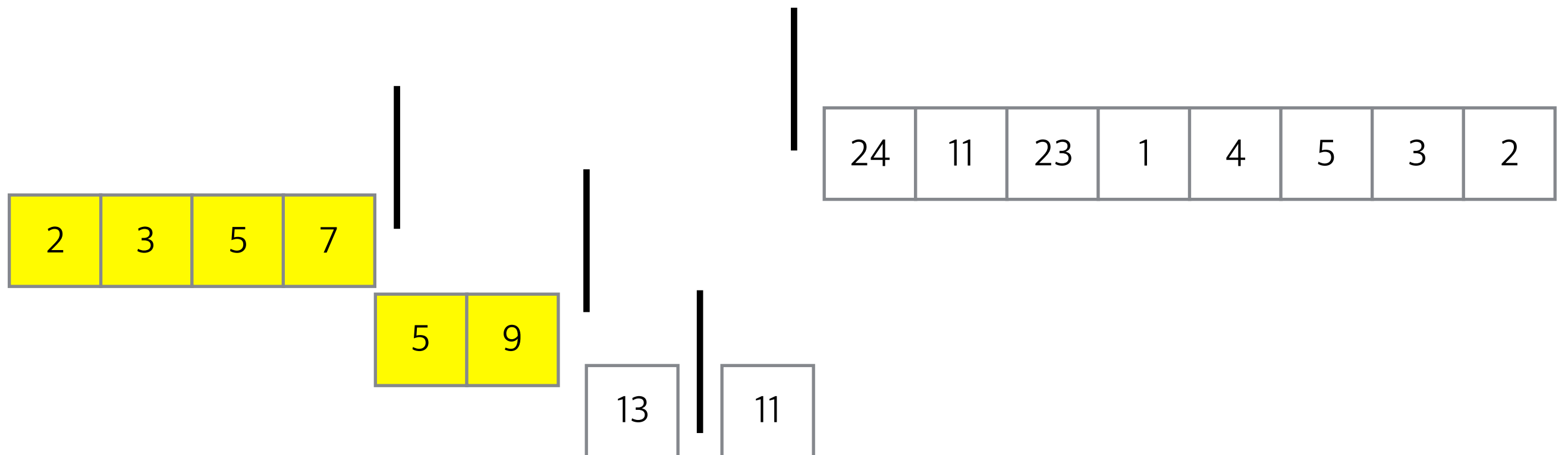
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



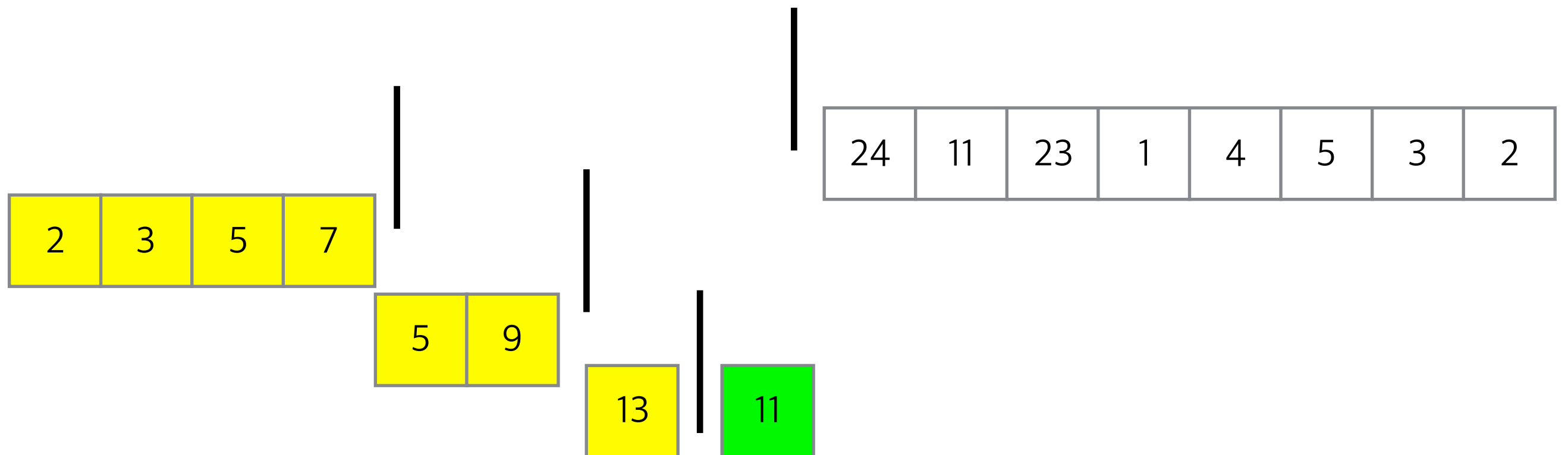
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



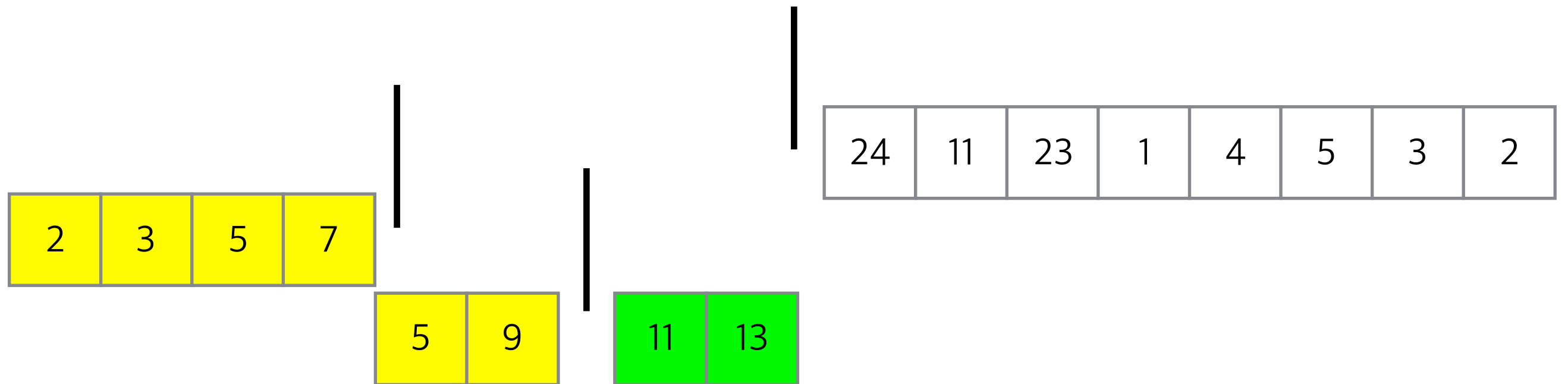
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



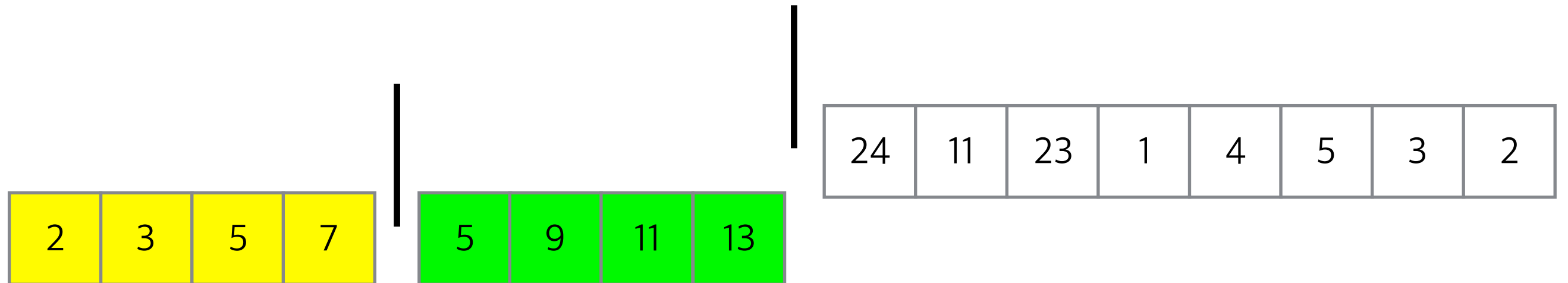
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



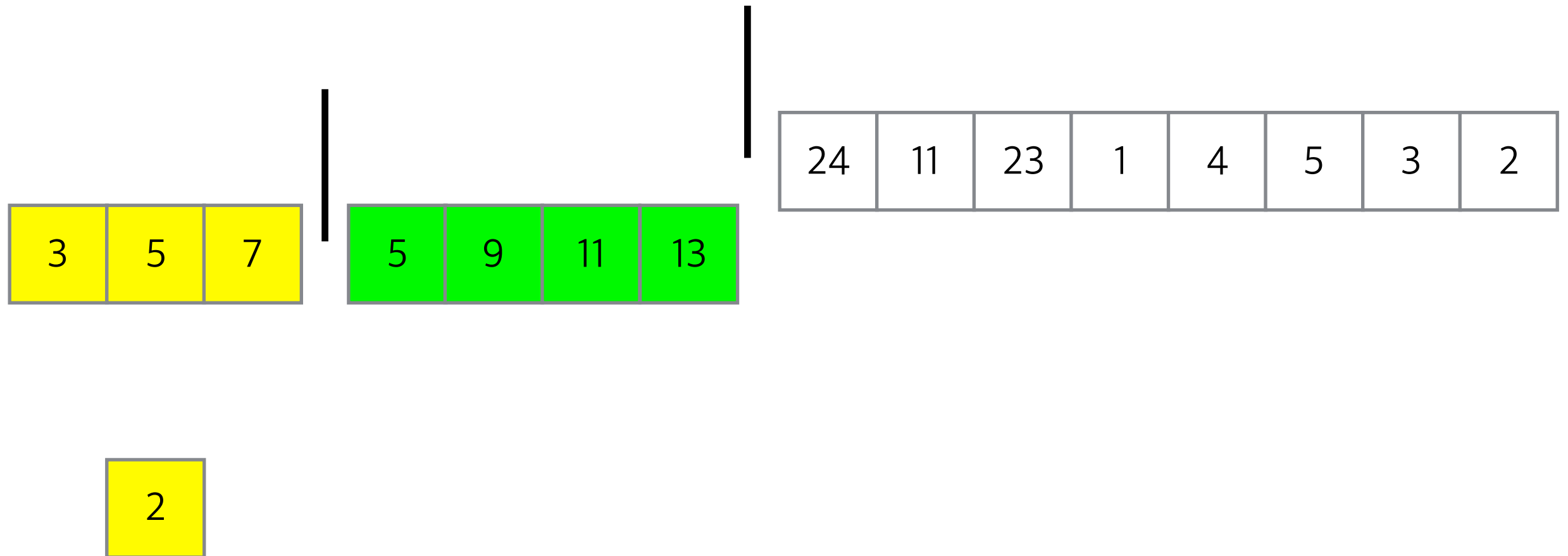
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



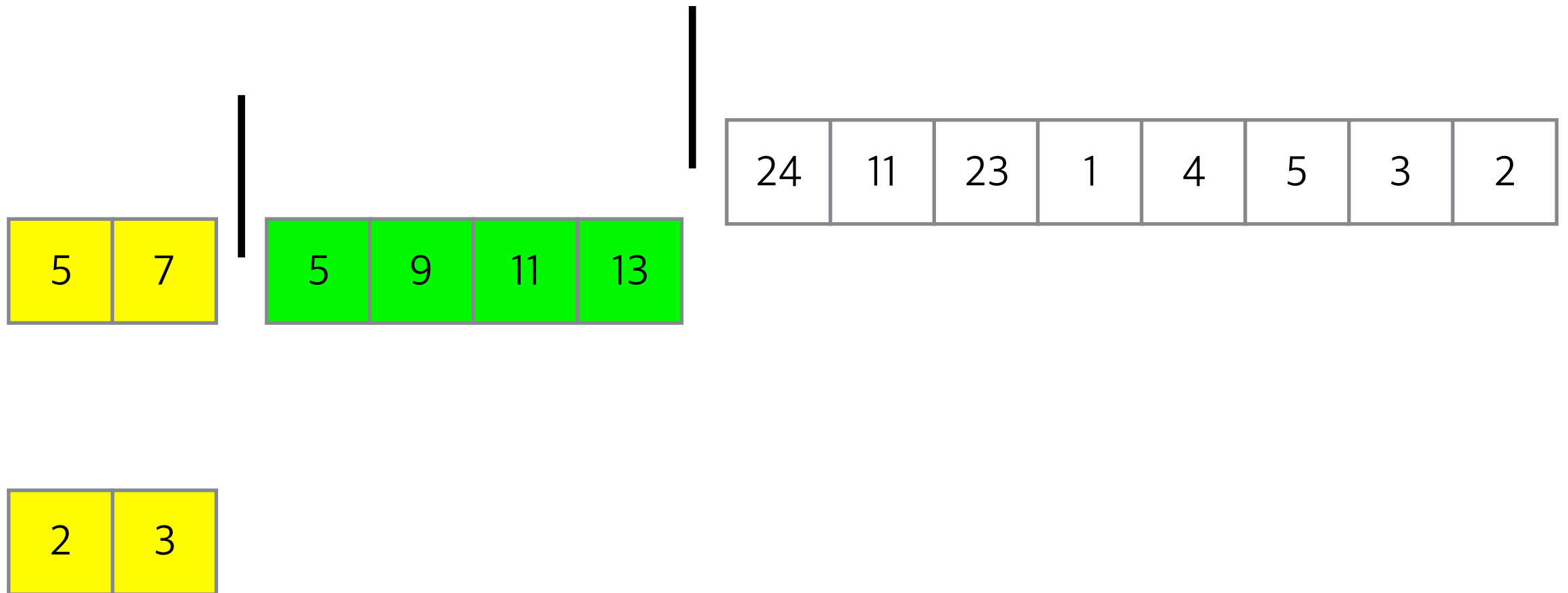
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



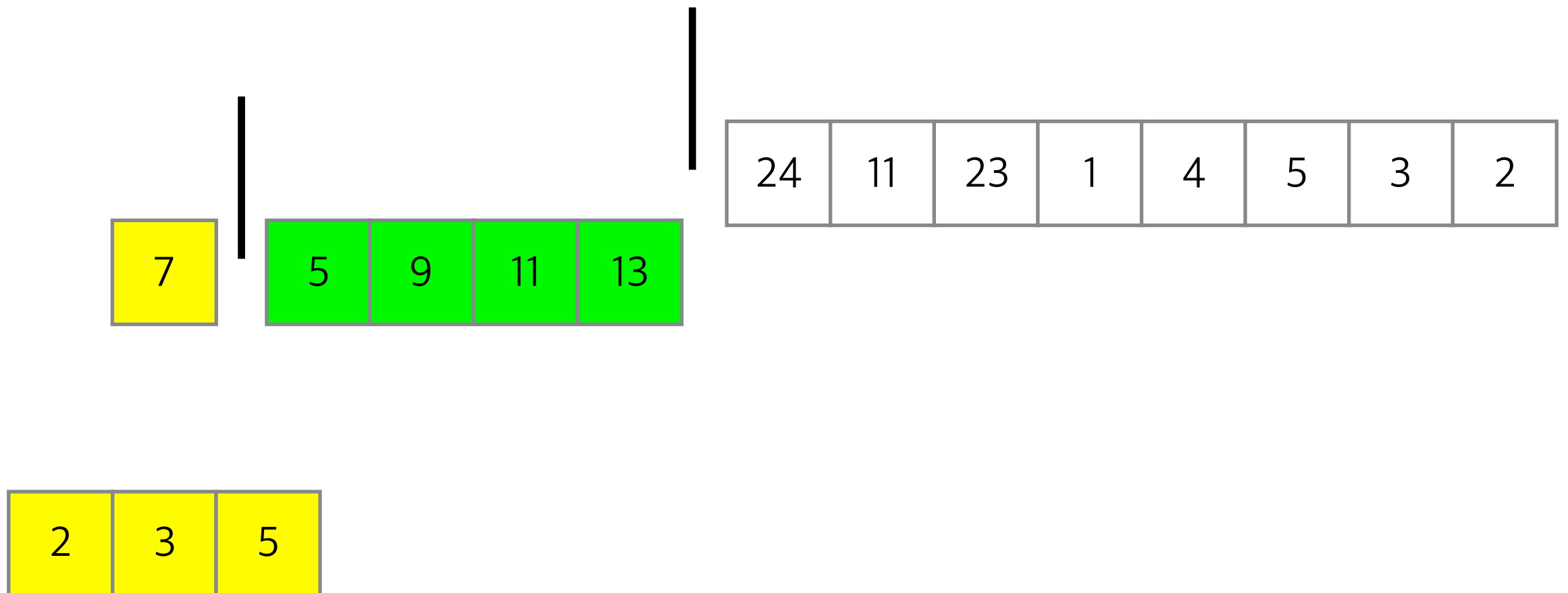
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



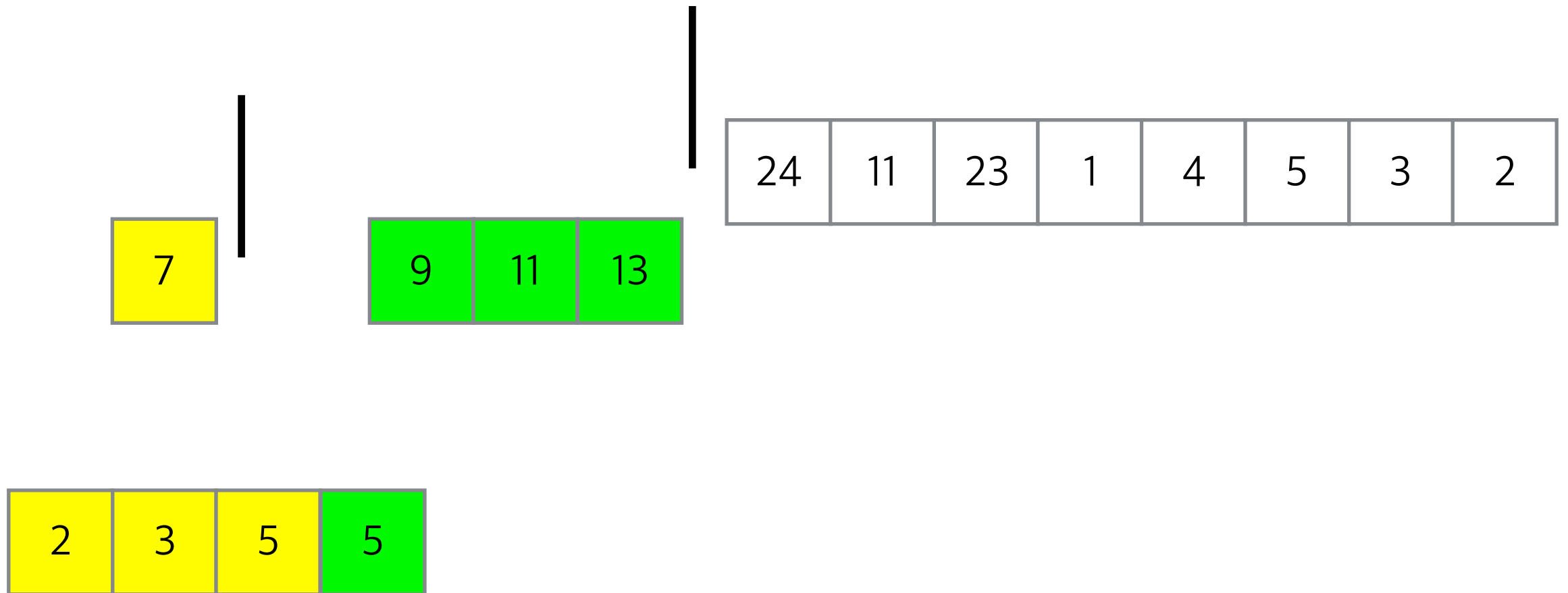
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



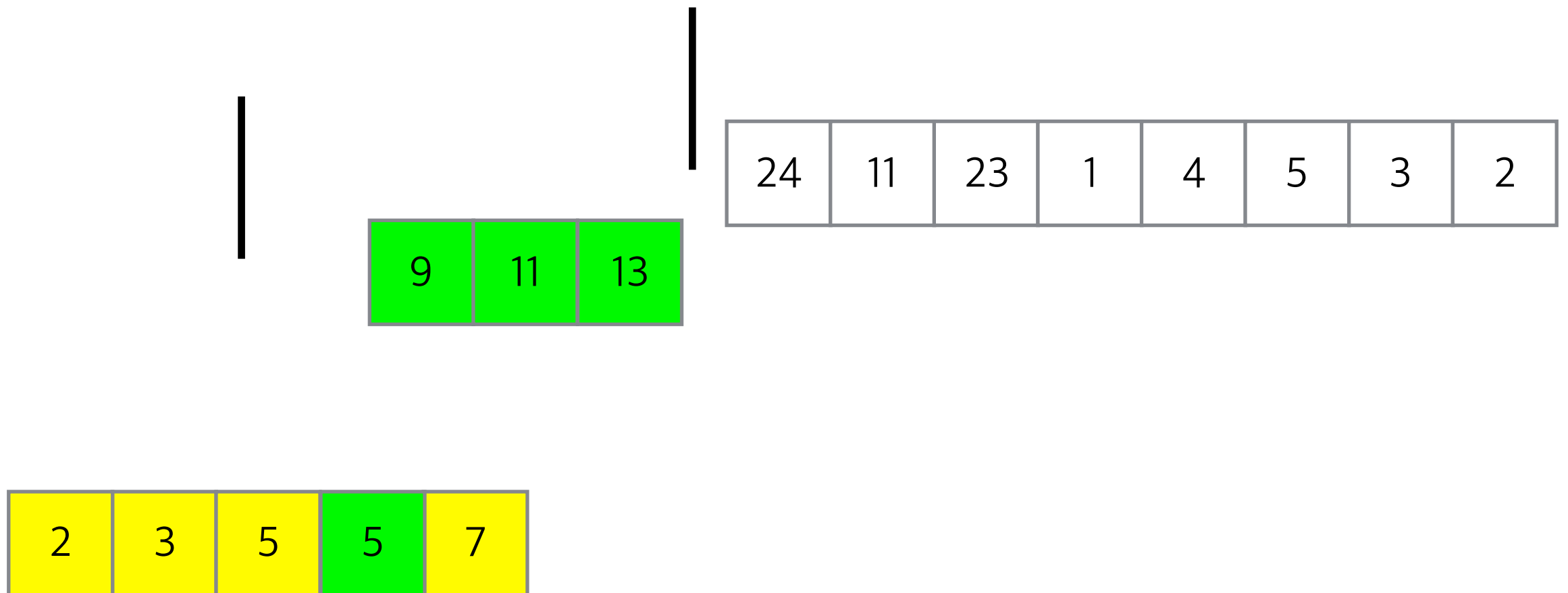
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



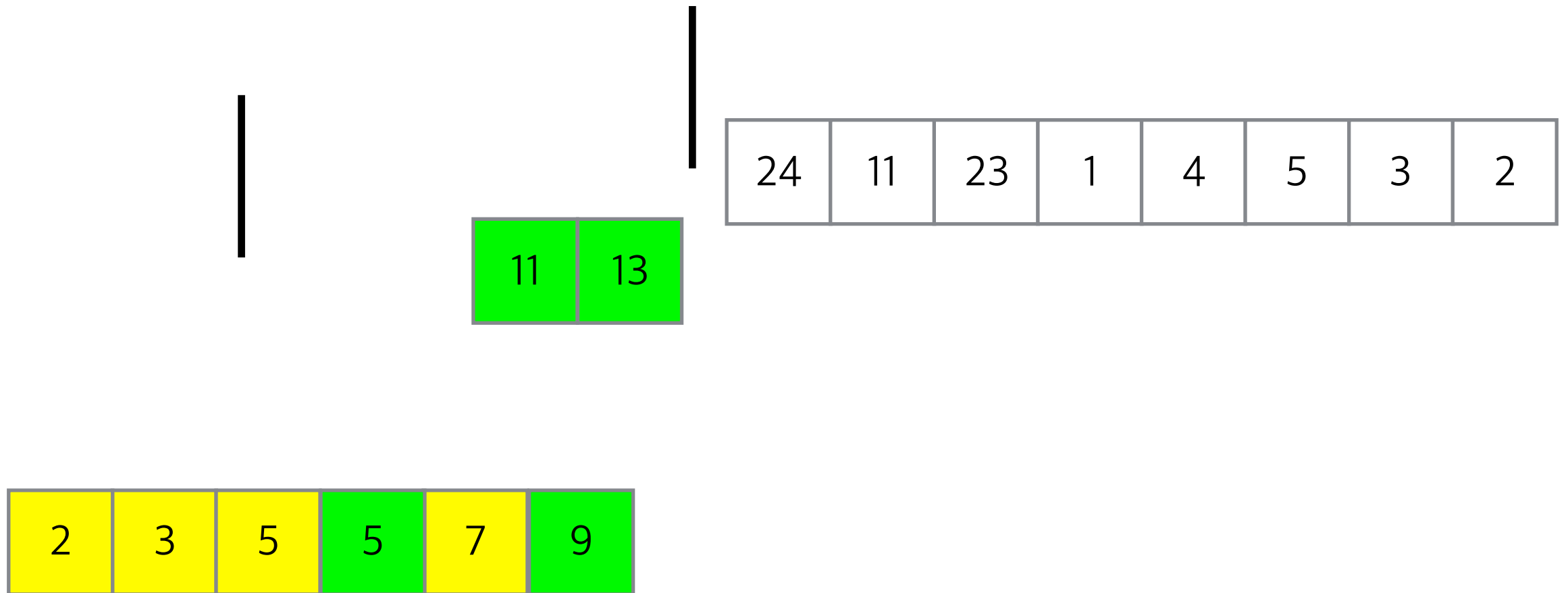
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



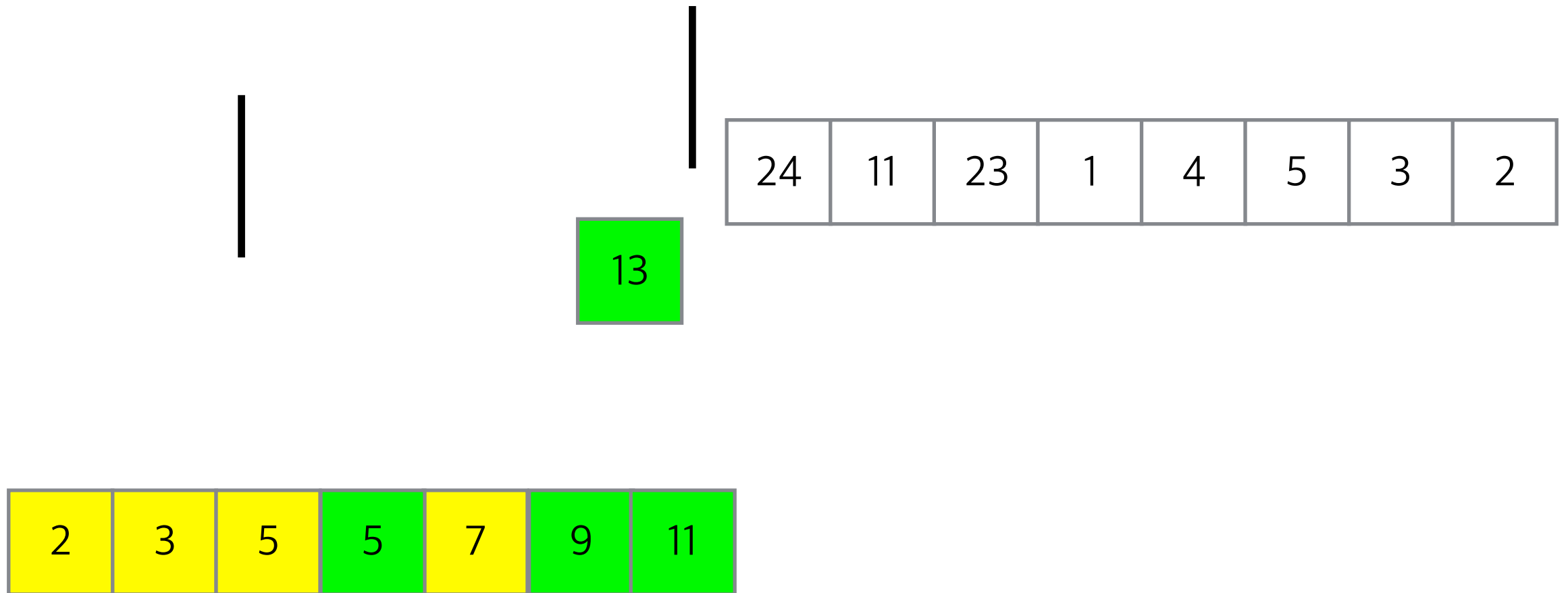
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



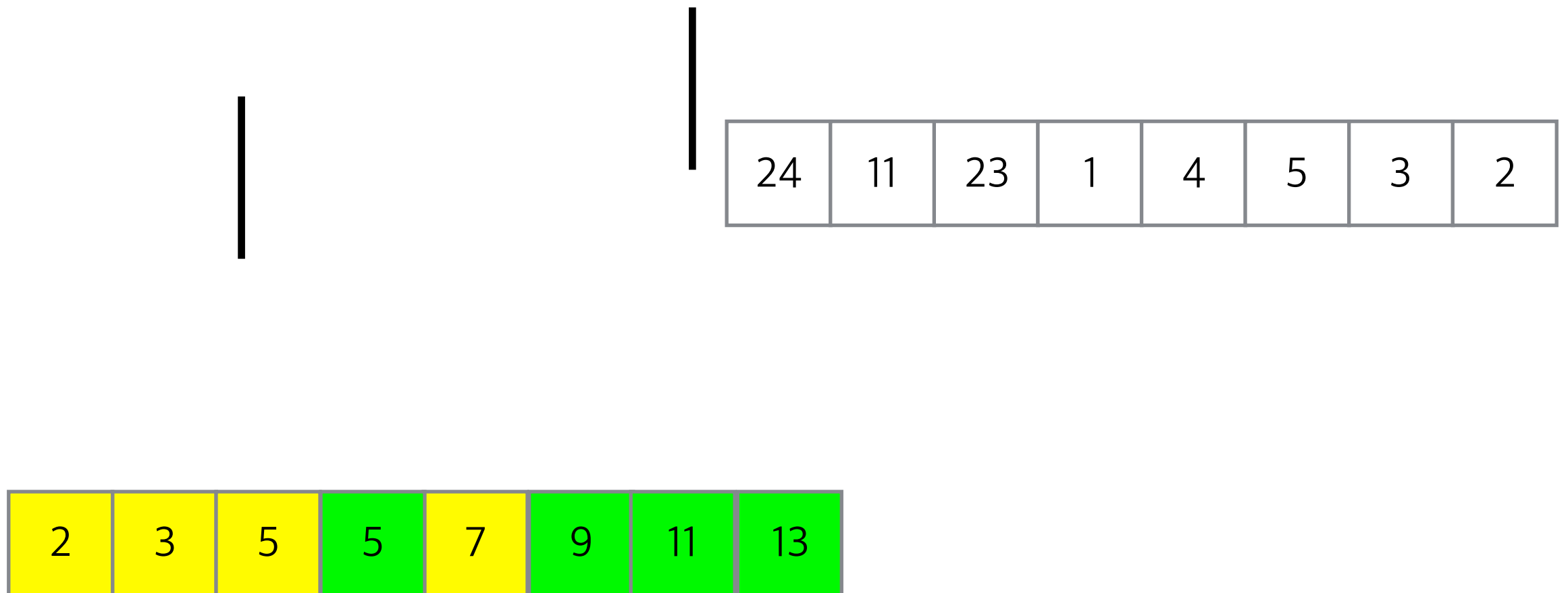
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



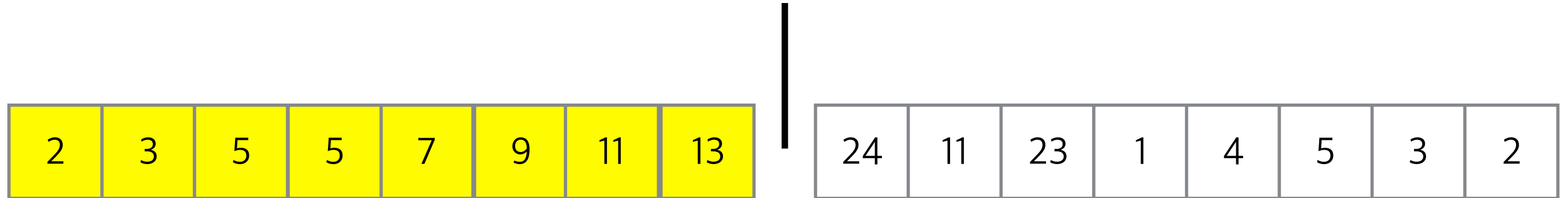
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



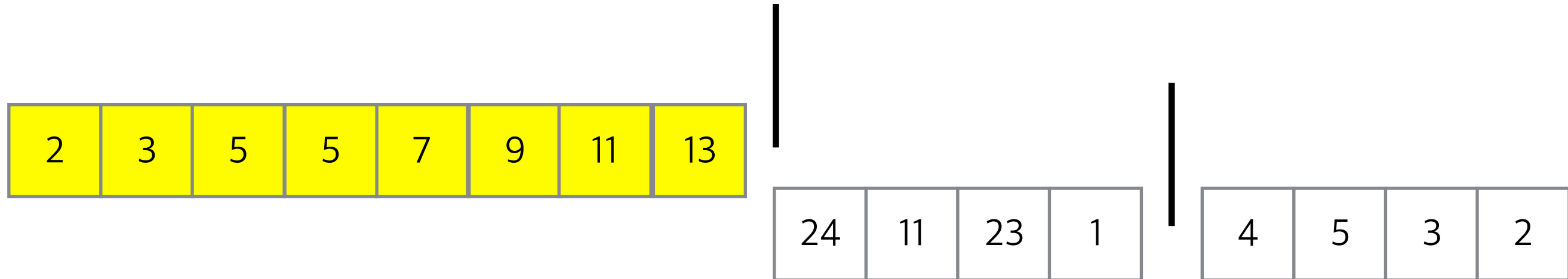
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



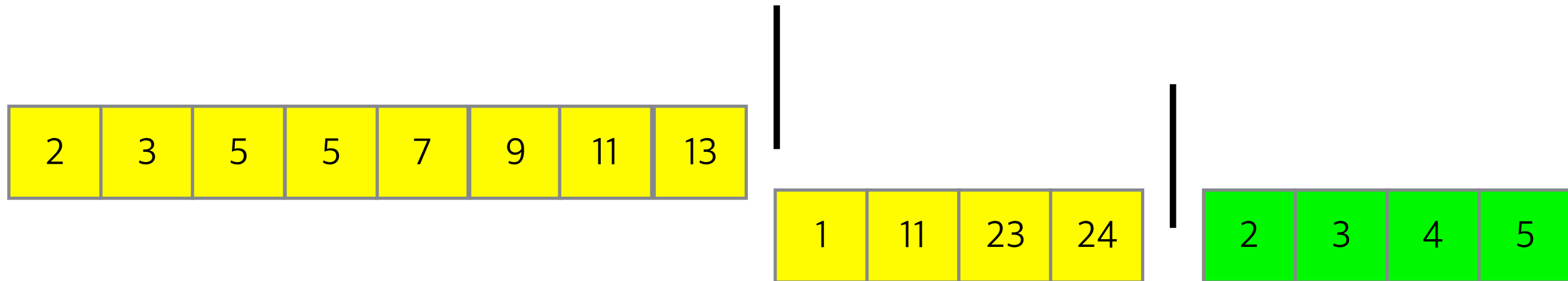
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



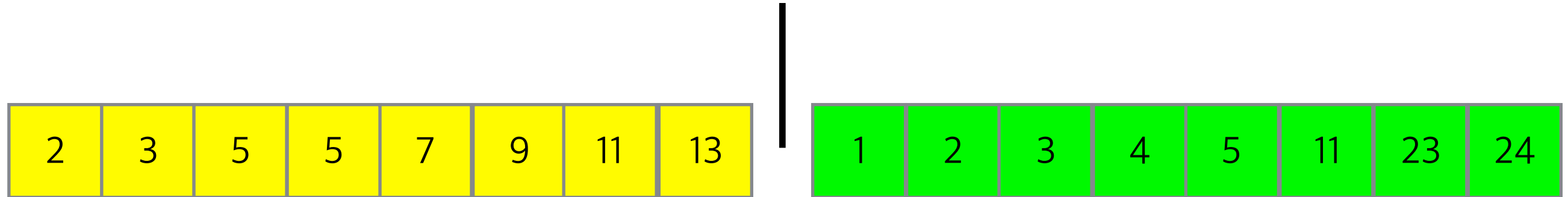
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



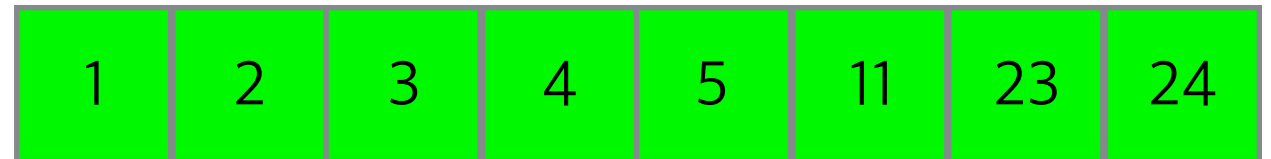
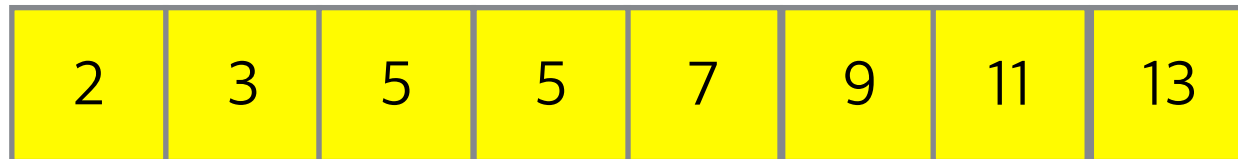
합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬



합병정렬 (Merge Sort)

재귀호출을 이용한 대표적인 정렬

1	2	2	3	3	4	5	5	5	7	9	11	11	13	23	24
---	---	---	---	---	---	---	---	---	---	---	----	----	----	----	----

합병 정렬의 시간복잡도

$T(N)$: N 개의 숫자를 정렬하는 데 걸리는 시간

합병 정렬의 시간복잡도

$T(N)$: N개의 숫자를 정렬하는 데 걸리는 시간

$$T(N) = 2 * T(N/2) + O(N)$$

합병 정렬의 시간복잡도

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

$$T(N) = 2 * T(N/2) + \mathbf{O(N)}$$

합병 정렬의 시간복잡도

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

$$T(N) = 2 * T(N/2) + O(N)$$

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

합병 정렬의 시간복잡도

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

$$T(N) = 2 * T(N/2) + O(N)$$

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

3	5	7	2
---	---	---	---

5	9	13	11
---	---	----	----

3	5	7	2
---	---	---	---

5	9	13	11
---	---	----	----

$$T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)}$$

합병 정렬의 시간복잡도

3	5	7	2	5	9	13	11	24	11	23	1	4	5	3	2
---	---	---	---	---	---	----	----	----	----	----	---	---	---	---	---

$$T(N) = 2 * T(N/2) + O(N)$$

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

3	5	7	2	5	9	13	11
---	---	---	---	---	---	----	----

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

$$T(N/2) = 2 * T(N/4) + \mathbf{O(N/2)}$$

3	5	7	2
---	---	---	---

5	9	13	11
---	---	----	----

3	5	7	2
---	---	---	---

5	9	13	11
---	---	----	----

$$T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)} \quad T(N/4) = \cdots + \mathbf{O(N/4)}$$

$$\mathbf{O(N \log N)}$$

[예제 3] 합병정렬 구현

합병정렬을 구현하라
단, $1 \leq n \leq 100,000$

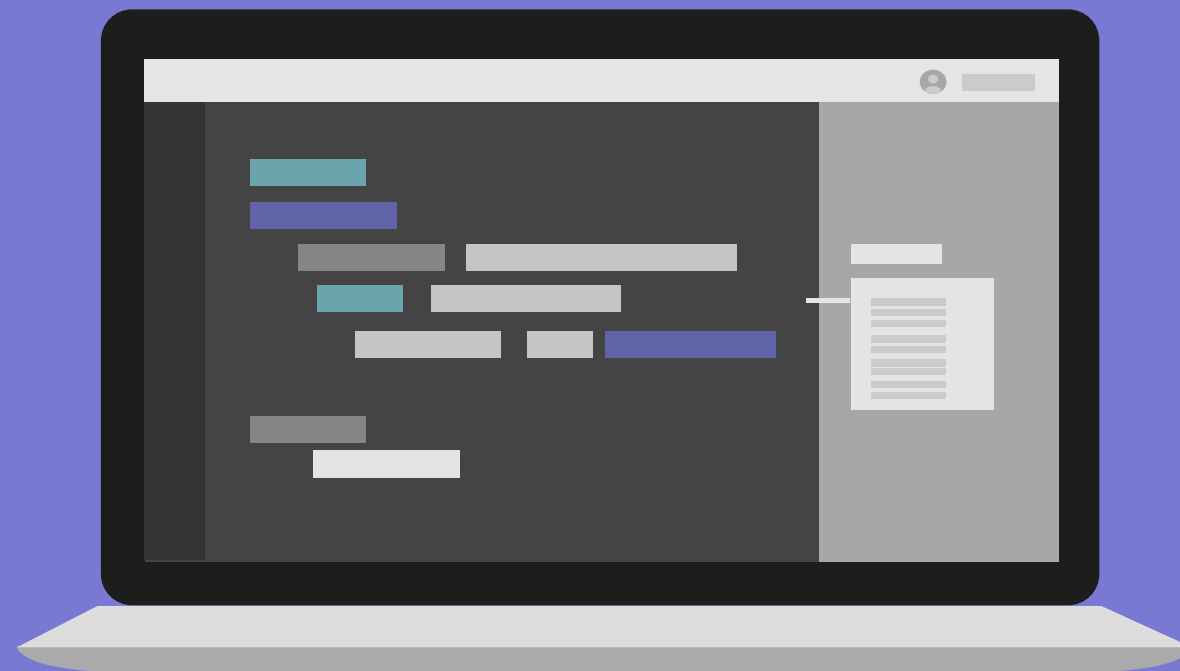
입력의 예

```
1 5 6 2 3 8 4 9 7 10
```

출력의 예

```
1 2 3 4 5 6 7 8 9 10
```

[예제 3] 합병정렬 구현



```
/* elice */
```

[예제 4] 연속부분 최대합 (Medium)

연속된 부분을 선택하였을 때, 그 최대 합을 출력

단, $1 \leq n \leq 100,000$

입력의 예

1 2 -4 5 3 -2 9 10

출력의 예

15

분할정복법 (Divide & Conquer)

문제를 소문제로 분할

각각의 소문제를 해결

소문제의 해결 결과를 이용하여 전체 문제를 해결

[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자

2	1	-2	5	-10	3	2	5	-3	7	9	-10
---	---	----	---	-----	---	---	---	----	---	---	-----

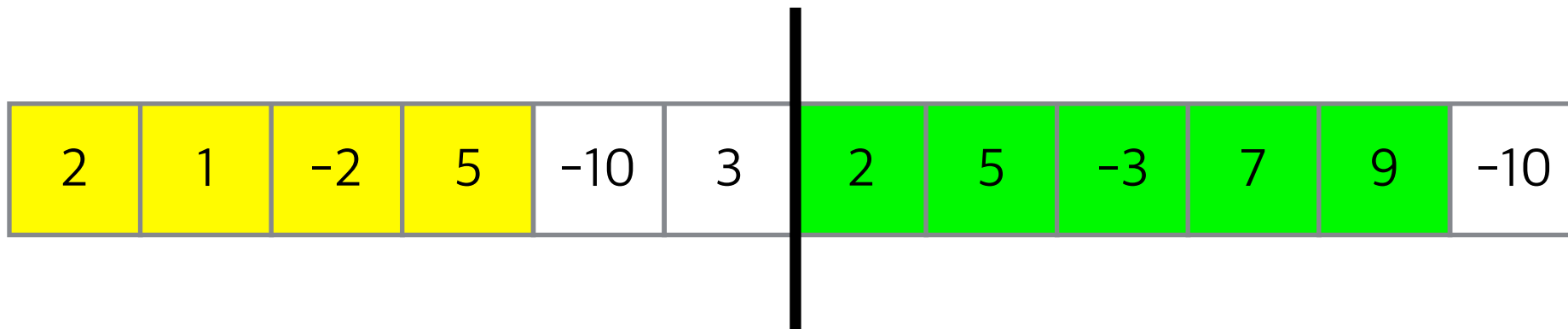
[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자

2	1	-2	5	-10	3	2	5	-3	7	9	-10
---	---	----	---	-----	---	---	---	----	---	---	-----

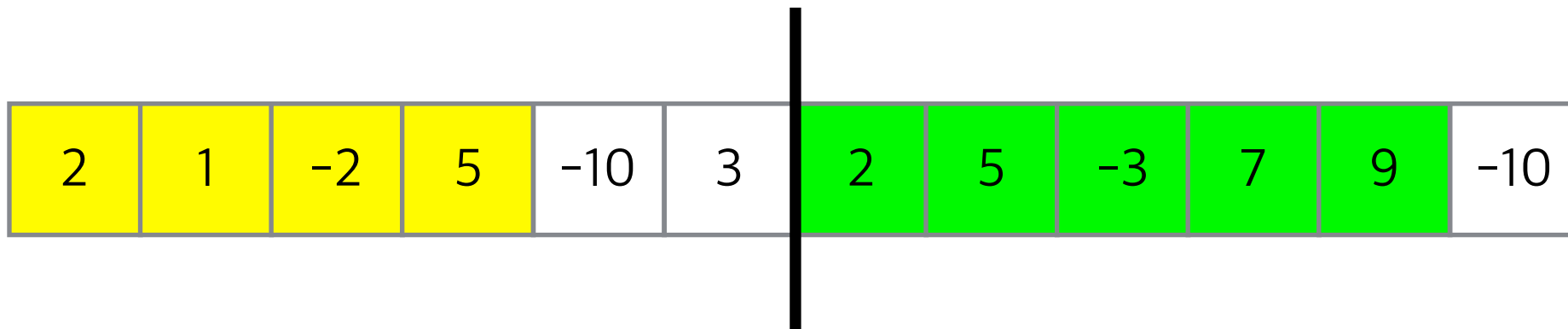
[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자



[예제 4] 연속부분 최대합 (Medium)

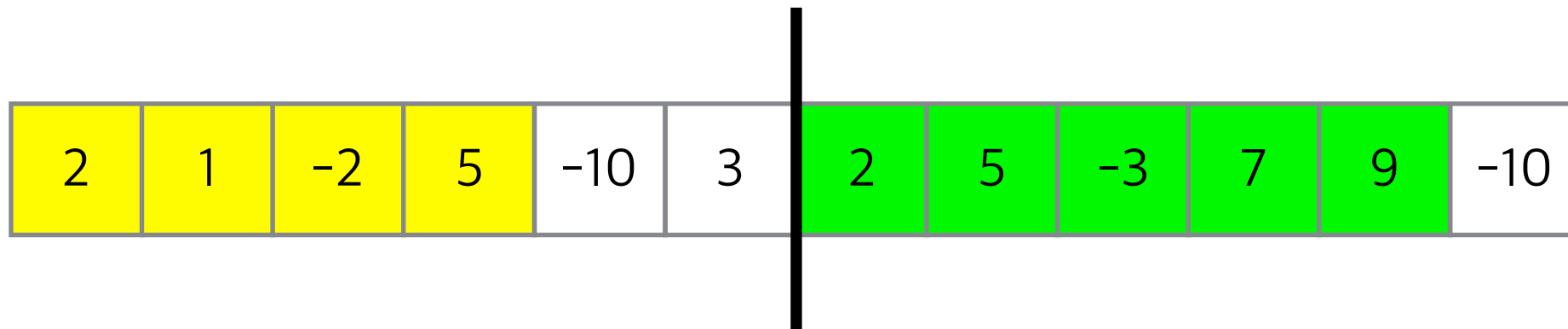
우선 절반으로 나누어 각각을 구해보자



고려하지 않은 경우는 ?

[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자

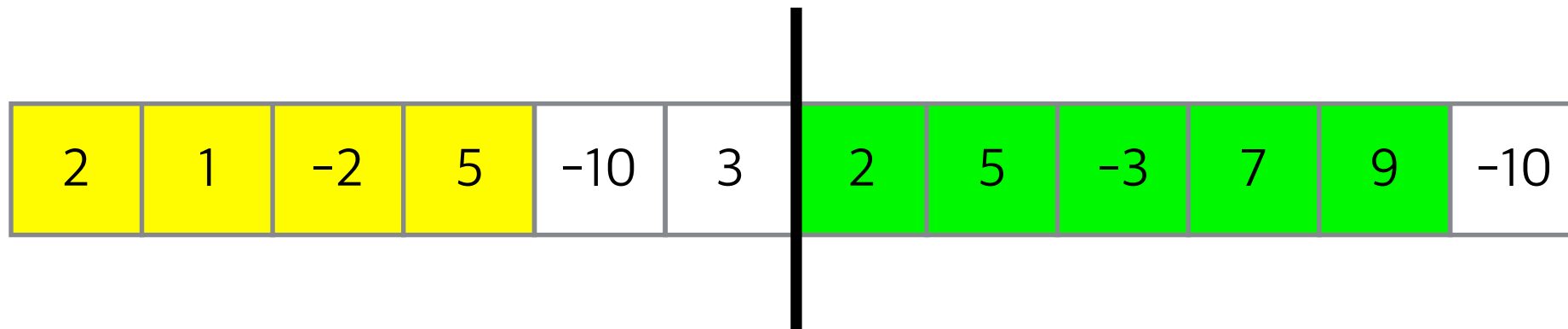


고려하지 않은 경우는 ?

자른 지점을 포함하는 연속 부분!

[예제 4] 연속부분 최대합 (Medium)

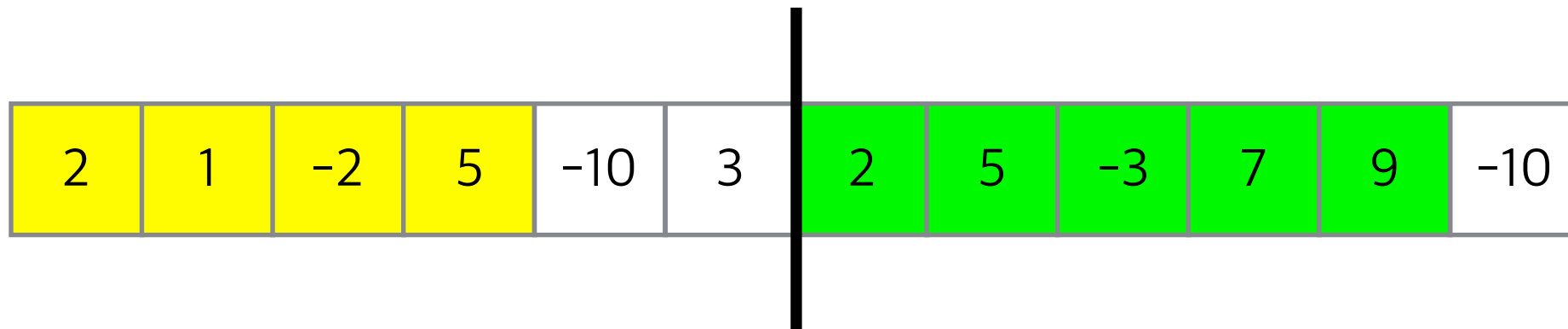
우선 절반으로 나누어 각각을 구해보자



자른 지점을 포함하는 연속 부분의 최대 합은 어떻게 구할까?

[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자

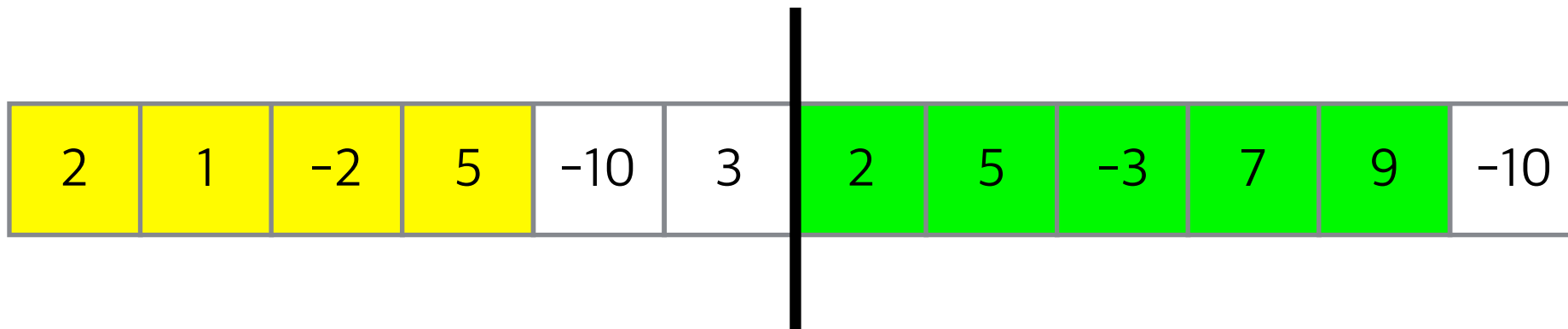


자른 지점을 포함하는 연속 부분의 최대 합은 어떻게 구할까?

Idea : 왼쪽과 오른쪽을 독립적으로 생각하자

[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자



[예제 4] 연속부분 최대합 (Medium)

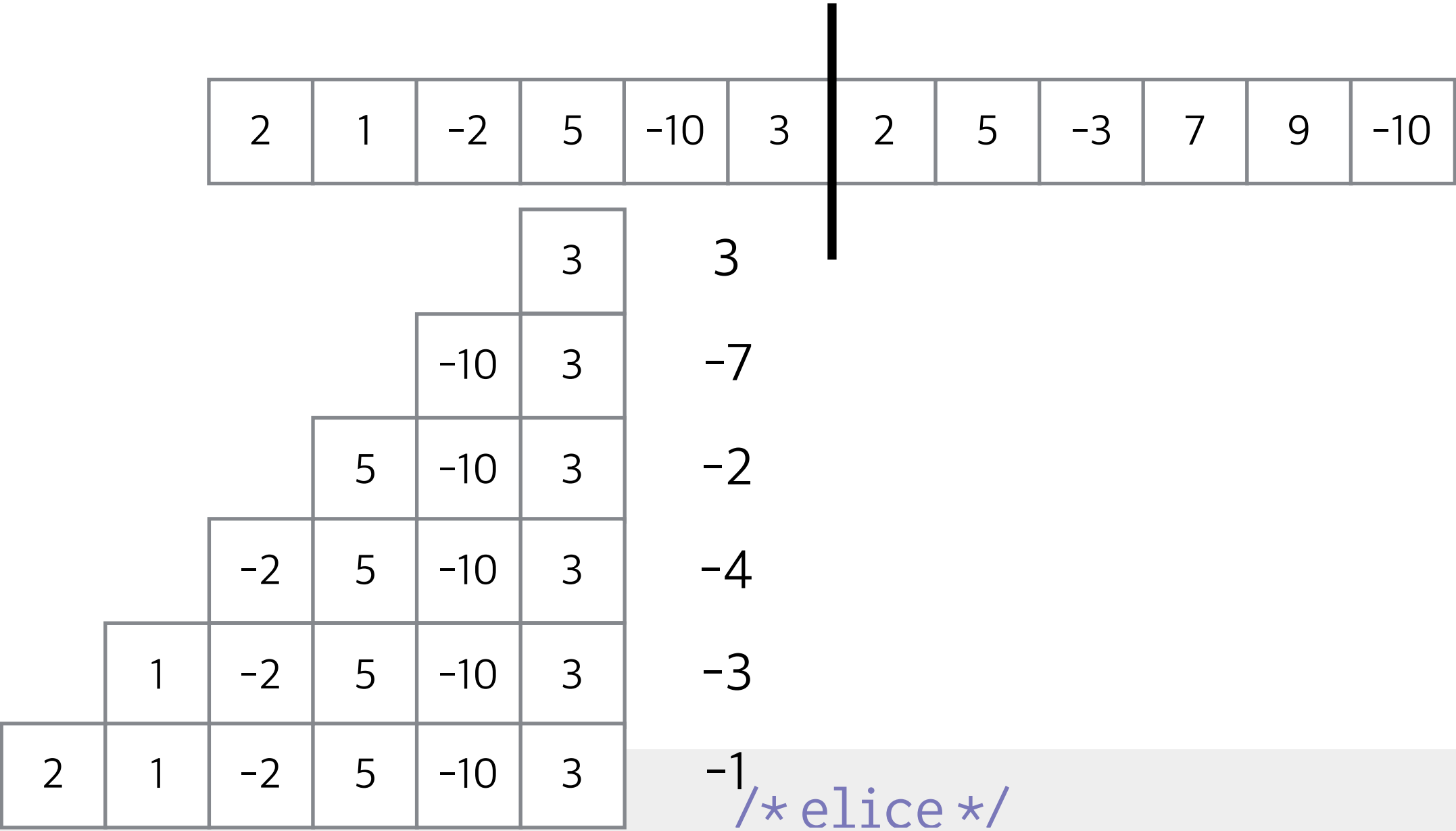
우선 절반으로 나누어 각각을 구해보자

2	1	-2	5	-10	3	2	5	-3	7	9	-10
---	---	----	---	-----	---	---	---	----	---	---	-----

					3
				-10	3
			5	-10	3
		-2	5	-10	3
	1	-2	5	-10	3
2	1	-2	5	-10	3

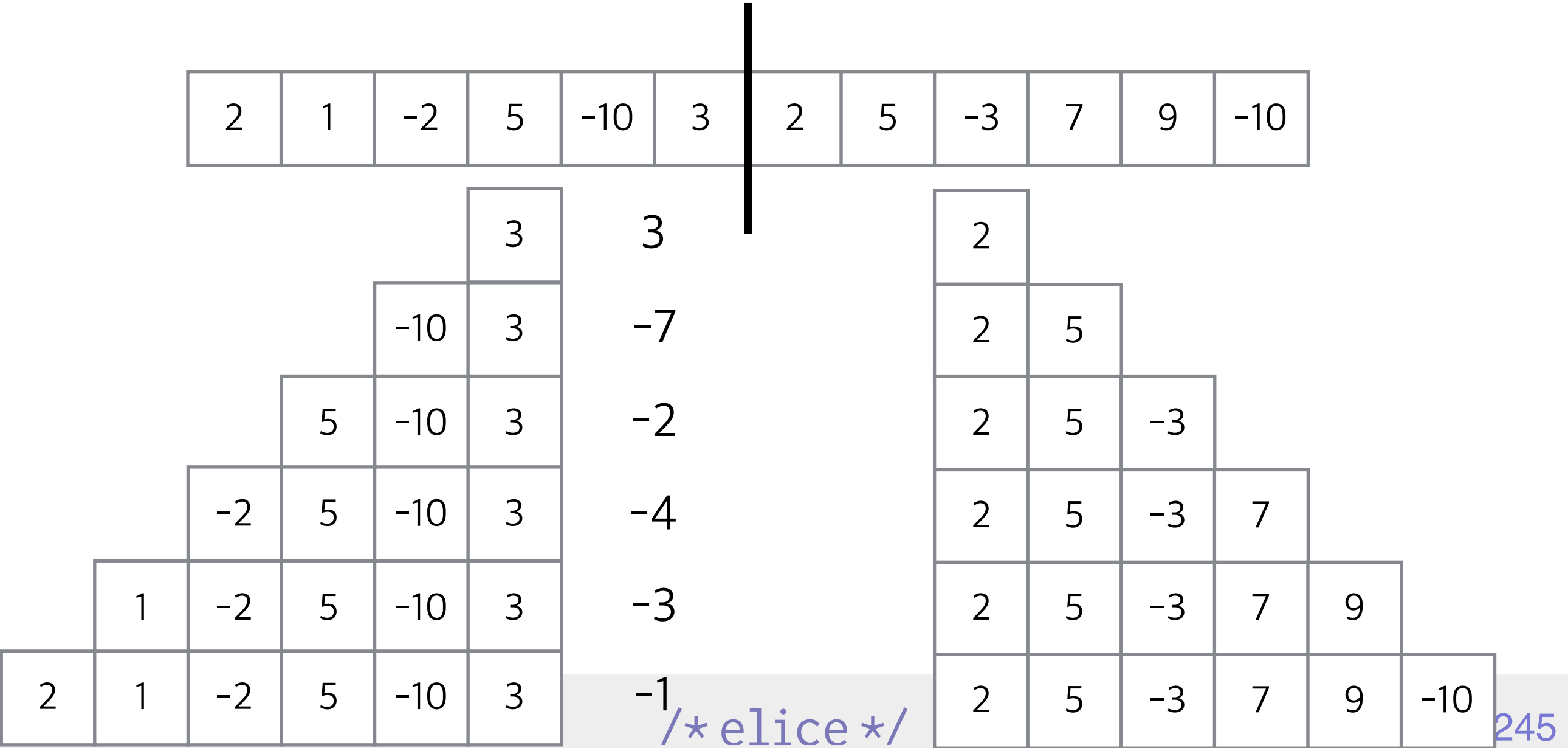
[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자



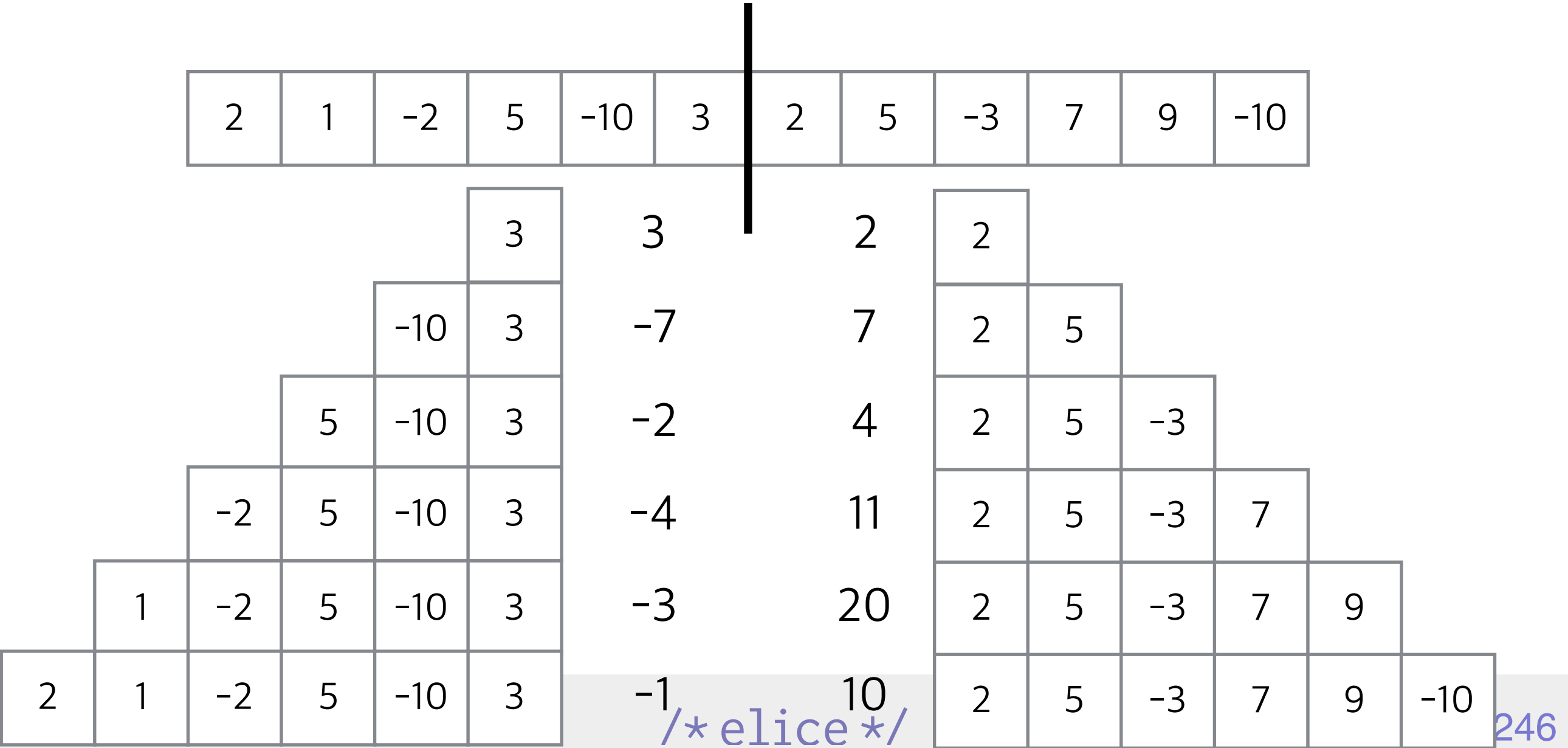
[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자



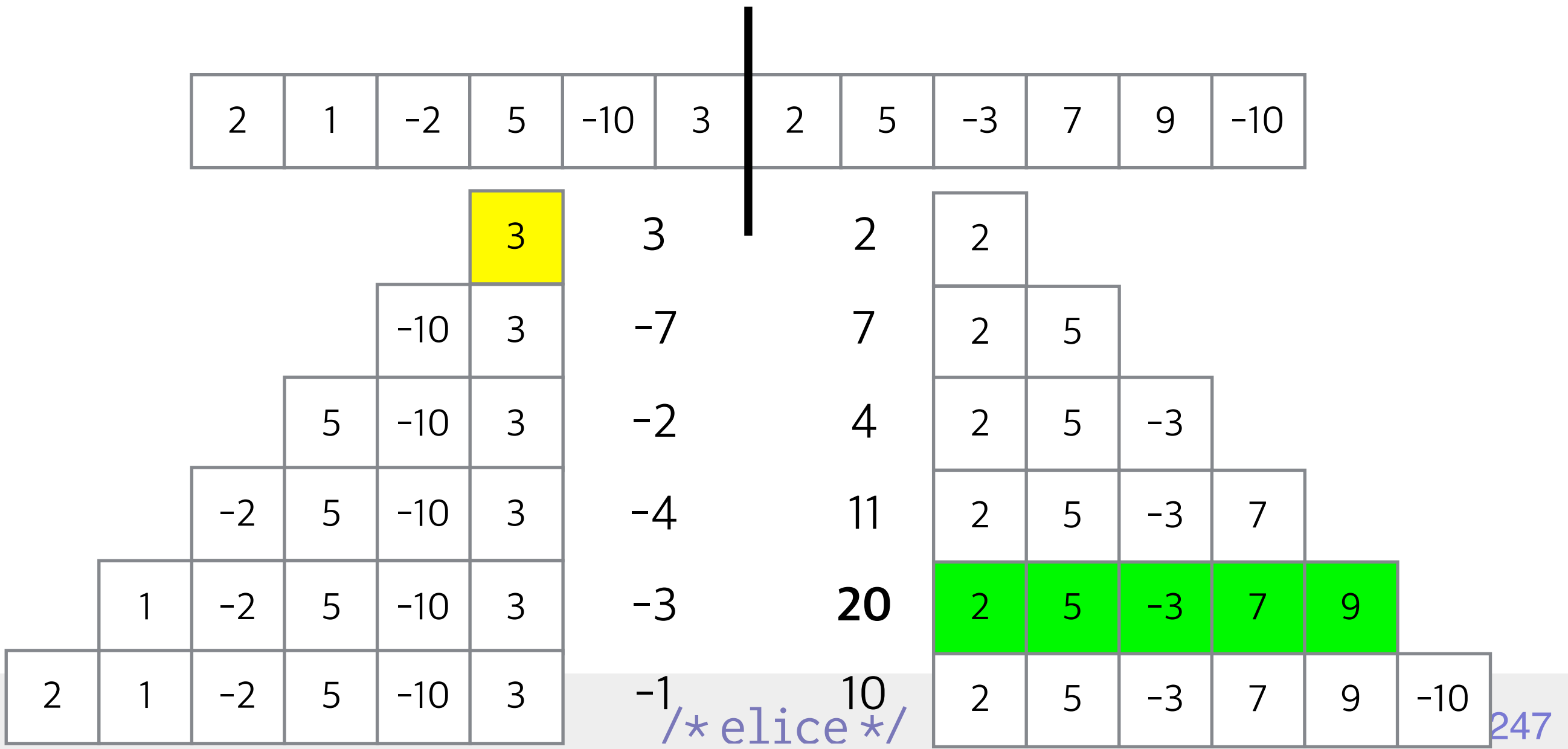
[예제 4] 연속부분 최대합 (Medium)

우선 절반으로 나누어 각각을 구해보자



[예제 4] 연속부분 최대합 (Medium)

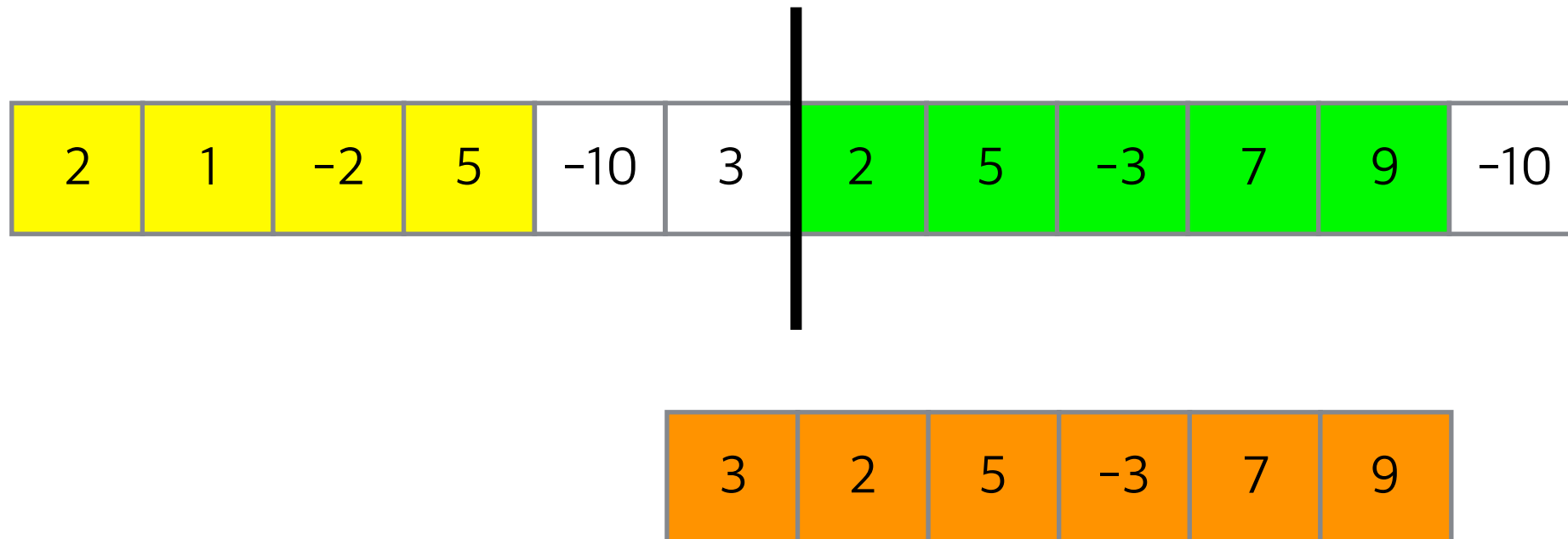
우선 절반으로 나누어 각각을 구해보자



[예제 4] 연속부분 최대합 (Medium)

모든 경우를 고려했음

1. 왼쪽만 포함하는 경우,
2. 오른쪽만 포함하는 경우,
3. 자른 자리를 포함하는 경우



[예제 4] 연속부분 최대합 (Medium)

최댓값!

2	1	-2	5	-10	3	2	5	-3	7	9	-10
---	---	----	---	-----	---	---	---	----	---	---	-----

[예제 4] 연속부분 최대합 (Medium)

시간복잡도

[예제 4] 연속부분 최대합 (Medium)

시간복잡도

$$T(N) = 2 * T(N/2) + O(N)$$

$$O(N \log N)$$

[예제 4] 연속부분 최대합 (Medium)



`/* elice */`

분할정복법 (Divide & Conquer)

문제를 소문제로 분할

각각의 소문제를 해결

소문제의 해결 결과를 이용하여 전체 문제를 해결

분할정복법 (Divide & Conquer)

어렵다

분할정복법 (Divide & Conquer)

어렵다

분할정복법으로 해결할 수 있는 문제인지

분할정복법 (Divide & Conquer)

어렵다

분할정복법으로 해결할 수 있는 문제인지
분할을 한다면, 어떻게 해야 하는지

분할정복법 (Divide & Conquer)

어렵다

분할정복법으로 해결할 수 있는 문제인지
분할을 한다면, 어떻게 해야 하는지
합칠때는 어떻게 합쳐야 하는지

분할정복법 (Divide & Conquer)

어렵다

분할정복법으로 해결할 수 있는 문제인지
분할을 한다면, 어떻게 해야 하는지
합칠때는 어떻게 합쳐야 하는지
코딩은 어떻게 하는지

분할정복법 (Divide & Conquer)

어렵다

분할정복법으로 해결할 수 있는 문제인지
분할을 한다면, 어떻게 해야 하는지
합칠때는 어떻게 합쳐야 하는지
코딩은 어떻게 하는지
빠르긴 빠르든지

...

분할정복법 (Divide & Conquer)

분할정복법으로 해결할 수 있는 대표적인 예제를 알아두자
합병정렬, 퀵정렬, 거듭제곱 구하기, 연속부분 최대합
[ADV] 히스토그램, Inversion counting, 가장 가까운 두 점 찾기

분할정복법 (Divide & Conquer)

분할정복법으로 해결할 수 있는 대표적인 예제를 알아두자
합병정렬, 퀵정렬, 거듭제곱 구하기, 연속부분 최대합
[ADV] 히스토그램, Inversion counting, 가장 가까운 두 점 찾기

수학적 문제 해결 능력이 **가장** 중요하다
코딩 능력 \neq 문제 해결 능력

분할정복법 (Divide & Conquer)

분할정복법으로 해결할 수 있는 대표적인 예제를 알아두자
합병정렬, 퀵정렬, 거듭제곱 구하기, 연속부분 최대합
[ADV] 히스토그램, Inversion counting, 가장 가까운 두 점 찾기

수학적 문제 해결 능력이 **가장** 중요하다
코딩 능력 \neq 문제 해결 능력

키보드 대신에 노트와 펜을 들고 생각하자

퀴즈 및 설문



/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

blog.naver.com/elicer