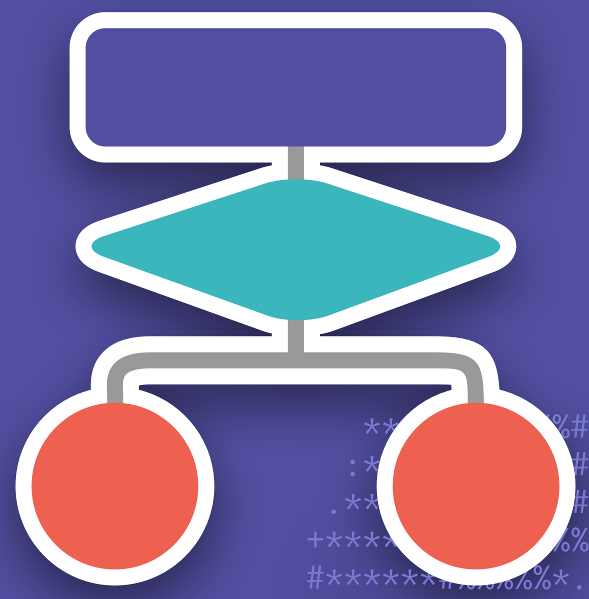
/* elice */

알고리즘의 정석

재귀호출



엘리스 선생님

커리큘럼

1 **재귀호출**

알고리즘을 이용한 문제 해결의 예를 알아보고, 문제 해결 기법의 근본적 이해를 위해 필수적인 개념인 재귀호출을 알아봅니다.

2 **문제 해결의 절차, 완전탐색, 시간복잡도**

주어진 문제에 대하여 가능한 모든 경우를 탐색하는 완전탐색법을 알아봅니다.

커리큘럼

3

분할정복법

주어진 문제를 작은 문제로 쪼개어, 각각의 작은 문제를 해결함으로써 전체 문제를 해결하는 분할정복법을 알아봅니다.

4

탐욕적 기법

항상 최적의 선택만을 함으로써 문제를 해결하는 탐욕적 기법을 알아봅니다.

수강 대상

코드의 성능을 끌어올리고 싶은 **개발자 혹은 국내외 IT 기업 기술 면접**을 준비하고 있는 분

ACM-ICPC 등 **각종 프로그래밍 경진대회**에 도전하고 싶으신 분

알고리즘 문제 해결을 체계적으로 배워본 경험이 부족하신 분

수강 목표

알고리즘이 무엇인지 알아봅니다.

알고리즘에서 제일 중요한 **재귀호출**에 대해서 알아봅니다.

재귀호출의 제일 대표적인 **퀵정렬**을 배워봅니다.

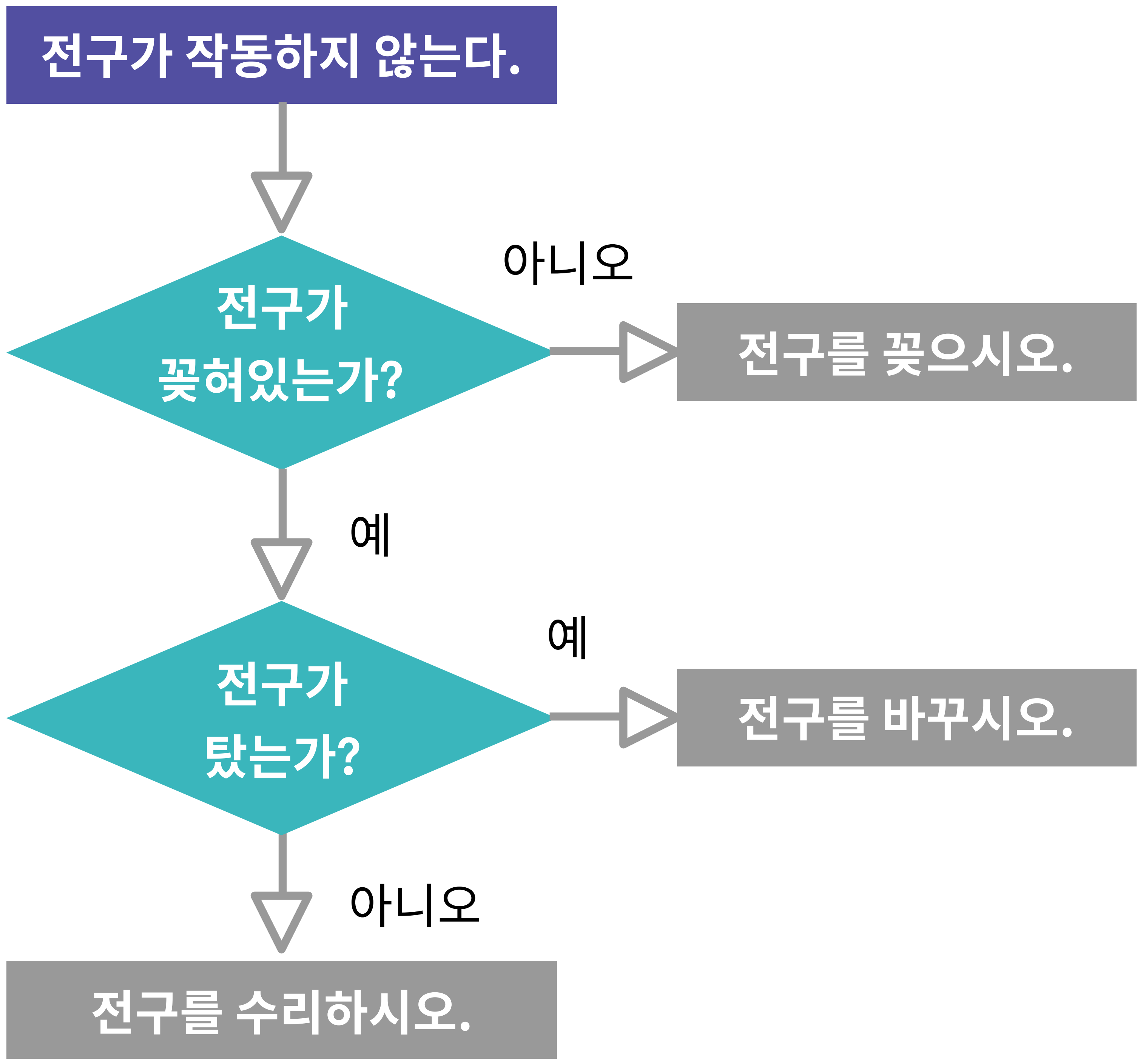
목차

1. 알고리즘이란?
2. 재귀호출
3. 수학적 귀납법
4. 퀵정렬
5. 재귀함수 디자인
6. 요약

알고리즘이란?

알고리즘

문제를 해결하는 방법




```python
def fixBulb(bulb) :
    if not bulb.isEmpty() :
        bulb.create()

    elif bulb.isBurnt :
        bulb.change()

    else :
        bulb.fix()
```

알고리즘

계산을 통하여 해결할 수 있는 문제를

해결하는 방법

[실습1] 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

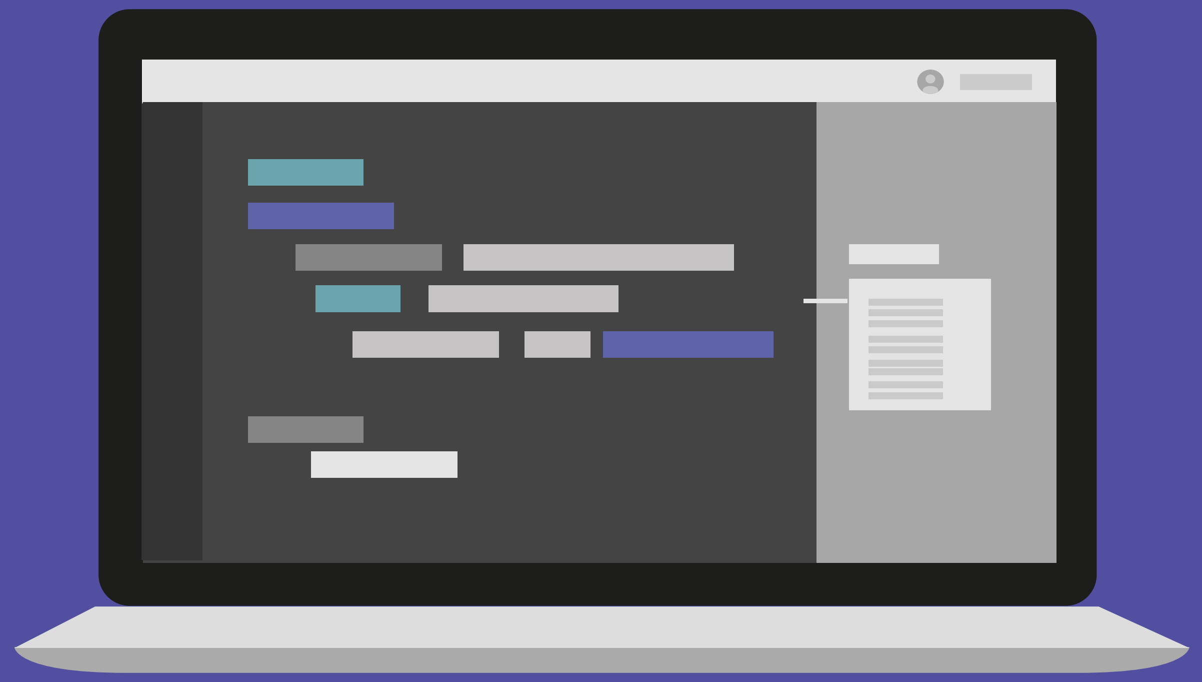
```
-1 -1 9 8 5 3 3 3 2 2
```

[실습1] k번째 숫자 찾기

문제를 해결하는 방법

1. 숫자 하나를 입력받는다.
2. 지금까지 받은 숫자들을 정렬한다.
3. k번째로 작은 숫자를 출력한다.

[실습 1] k번째 숫자 찾기



재귀호출

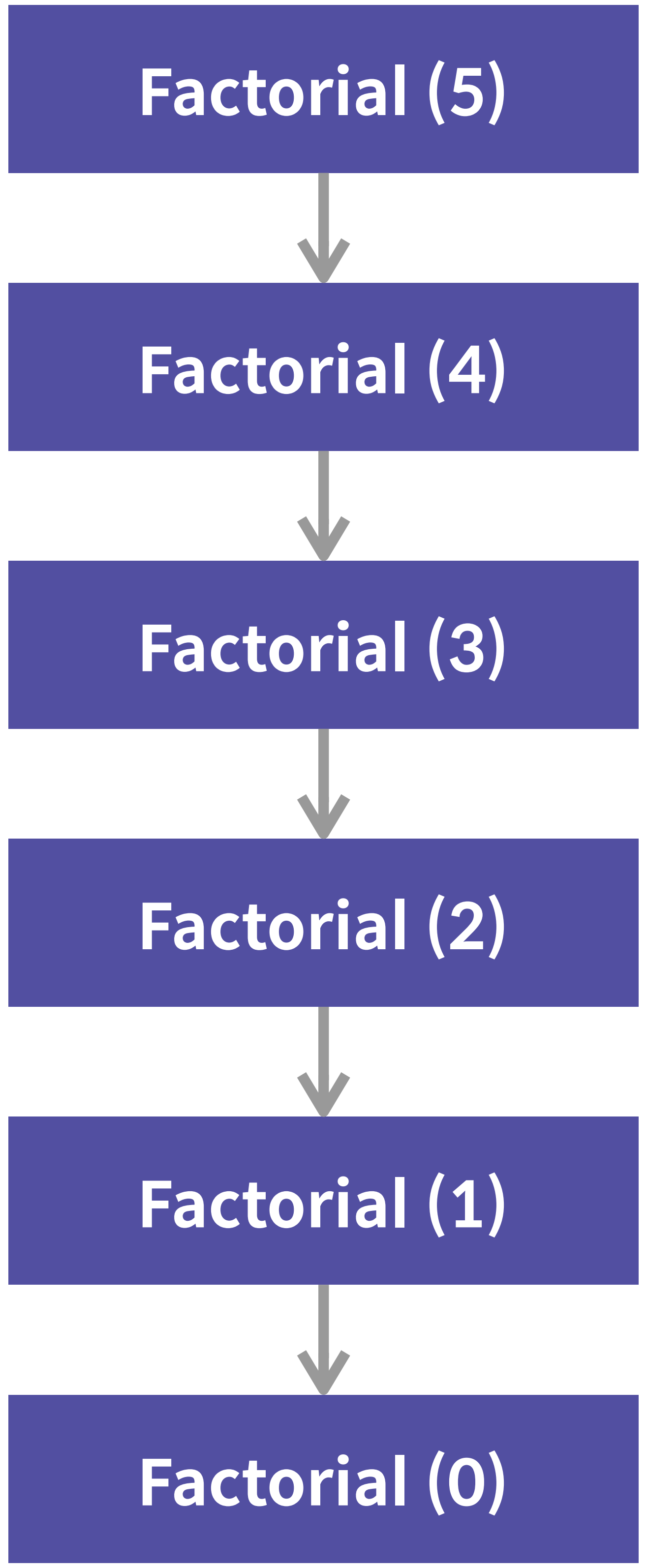
재귀호출

함수가 **자기 자신**을 호출

왜?

```
def Factorial(n) :
    if n == 0 :
        return 1
    else :
        return n * Factorial(n-1)
```

팩토리얼의 재귀적 정의



$$n! = n \times (n-1)!$$

Factorial(n) : n!을 반환하는 함수

```
def Factorial(n) :
    if n == 0 :
        return 1
    else :
        return n * Factorial(n-1)
```

수학적 귀납법

수학적 귀납법 = 재귀적 증명법

수학적 귀납법

명제 **P(n)**을 다음과 같이 증명하는 방법

1. **N = 1** 일 때 성립함을 보인다.
2. **P(k)**가 성립한다고 가정할 때, **P(k+1)**이 성립함을 보인다.
3. 따라서 **모든 자연수 n**에 대하여 P(n)이 성립한다.

수학적 귀납법

모든 자연수 n에 대하여 **$n! \le n^n$** 임을 증명하시오

1. **N = 1** 일 때 성립함을 보인다.
2. **P(k)**가 성립한다고 가정할 때, **P(k+1)**이 성립함을 보인다.
3. 따라서 **모든 자연수 n**에 대하여 P(n)이 성립한다.

$$1! \le 1^1$$

수학적 귀납법

모든 자연수 n에 대하여 $n! \leq n^n$ 임을 증명하시오

1. **N = 1** 일 때 성립함을 보인다.
2. **P(k)**가 성립한다고 가정할 때, **P(k+1)**이 성립함을 보인다.
3. 따라서 **모든 자연수 n**에 대하여 P(n)이 성립한다.

$$k! \leq k^k$$

수학적 귀납법

모든 자연수 n에 대하여 **$n! \leq n^n$** 임을 증명하시오

1. **N = 1** 일 때 성립함을 보인다.
2. **P(k)**가 성립한다고 가정할 때, **P(k+1)**이 성립함을 보인다.
3. 따라서 **모든 자연수 n**에 대하여 P(n)이 성립한다.

$$k! \leq k^k$$

$$k! \times (k+1) \leq k^k \times (k+1)$$

$$(k+1)! \leq (k+1)^{k+1}$$

수학적 귀납법

모든 자연수 n에 대하여 **$n! \leq n^n$** 임을 증명하시오

↓

하지만 **P(k)**가 실제로 성립하는지는 **아직 모른다.**

수학적 귀납법

모든 자연수 n에 대하여 **$n! \leq n^n$** 임을 증명하시오

n = 1	n = 2	n = 3	n = 4	n = 5	n = 6
$1! \leq 1^1$	$2! \leq 2^2$	$3! \leq 3^3$	$4! \leq 4^4$	$5! \leq 5^5$	$6! \leq 6^6$

수학적 귀납법

모든 자연수 n에 대하여 $\mathbf{n! \leq n^n}$ 임을 증명하시오

↓

따라서 **모든 자연수 n**에 대하여 위의 **명제가 성립한다.**

수학적 귀납법

명제 **P(n)**을 다음과 같이 증명하는 방법

1. **N = 1** 일 때 성립함을 보인다.
2. **P(k)**가 성립한다고 가정할 때, **P(k+1)**이 성립함을 보인다.
3. 따라서 **모든 자연수 n**에 대하여 P(n)이 성립한다.

수학적 귀납법을 거꾸로 보기

$n! \le n^n$ 이기 위해서는 **$(n-1)! \le (n-1)^{n-1}$** 이어야 한다.

n = 1일 경우에는 명제가 **성립한다.**

n = 1	n = 2	n = 3	n = 4	n = 5	n = 6
$1! \le 1^1$	$2! \le 2^2$	$3! \le 3^3$	$4! \le 4^4$	$5! \le 5^5$	$6! \le 6^6$

수학적 귀납법 = 재귀적 증명법

재귀호출 = 재귀적 계산법

재귀적 계산 방법

Factorial(n) : n!을 반환하는 함수

Factorial(n) = Factorial(n-1) * n

n=1 일때는 **Factorial(n)**이 정상 작동한다.

n = 1	n = 2	n = 3	n = 4	n = 5	n = 6
1! =	2! =	3! =	4! =	5! =	**6! =**

재귀적 계산 방법

Factorial(n) : n!을 반환하는 함수

```
def Factorial(n) :
    if n == 0 :
        return 1
    else :
        return n * Factorial(n-1)
```

퀵정렬(Quick Sort)

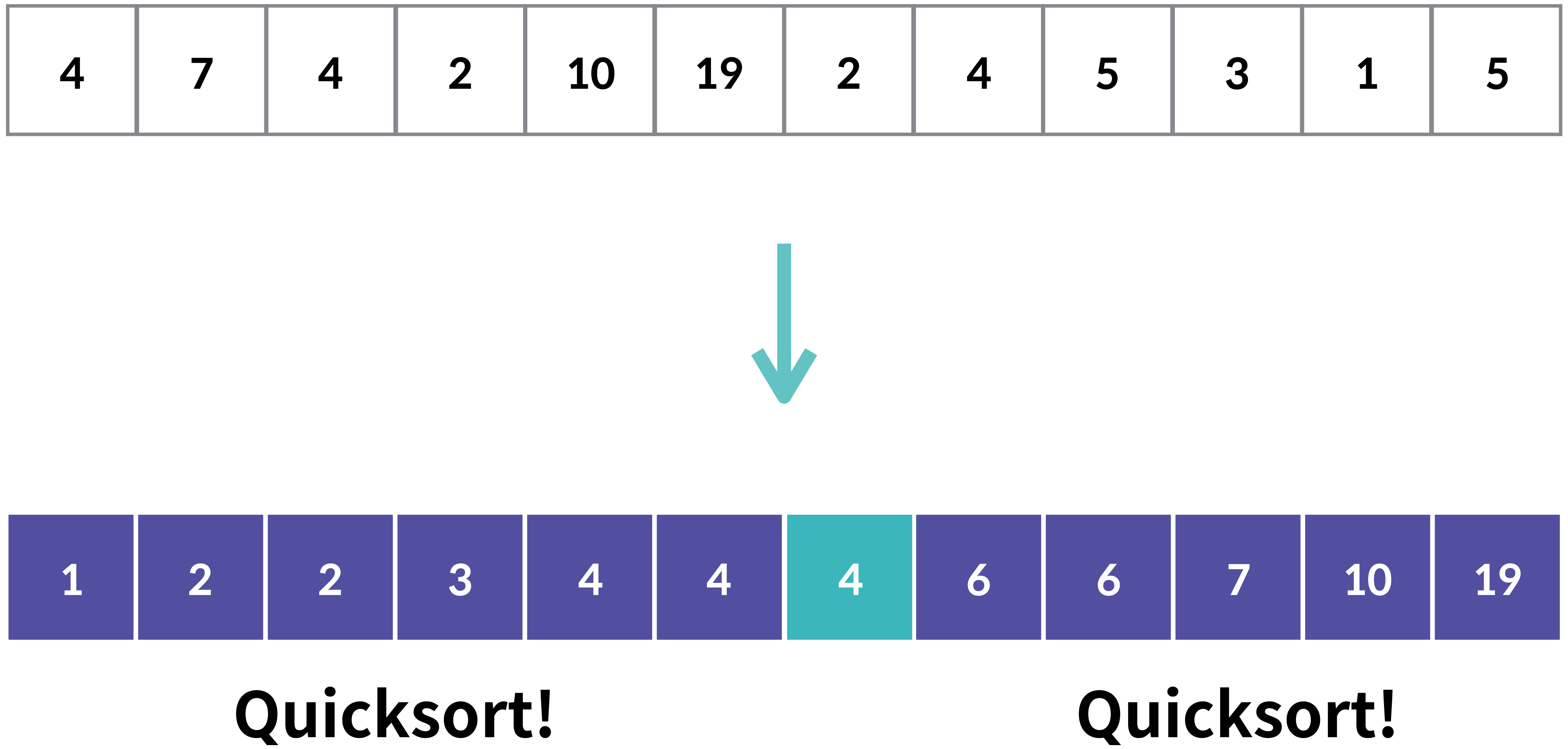
퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬

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

퀵정렬 (Quick Sort)

재귀호출을 이용한 대표적인 정렬



퀵정렬 (Quick Sort)

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

퀵정렬 (Quick Sort)

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

```python
def quickSort(data) :
    if len(data) <= 1 :
        return data

pivot = data[0]

left = getSmallNumbers(data, pivot)
right = getLargeNumbers(data, pivot)

return quickSort(left) + [pivot] + quickSort(right)
```

[실습2] 퀵정렬 구현하기

재귀호출의 대표적인 정렬 **퀵정렬**을 구현해본다.

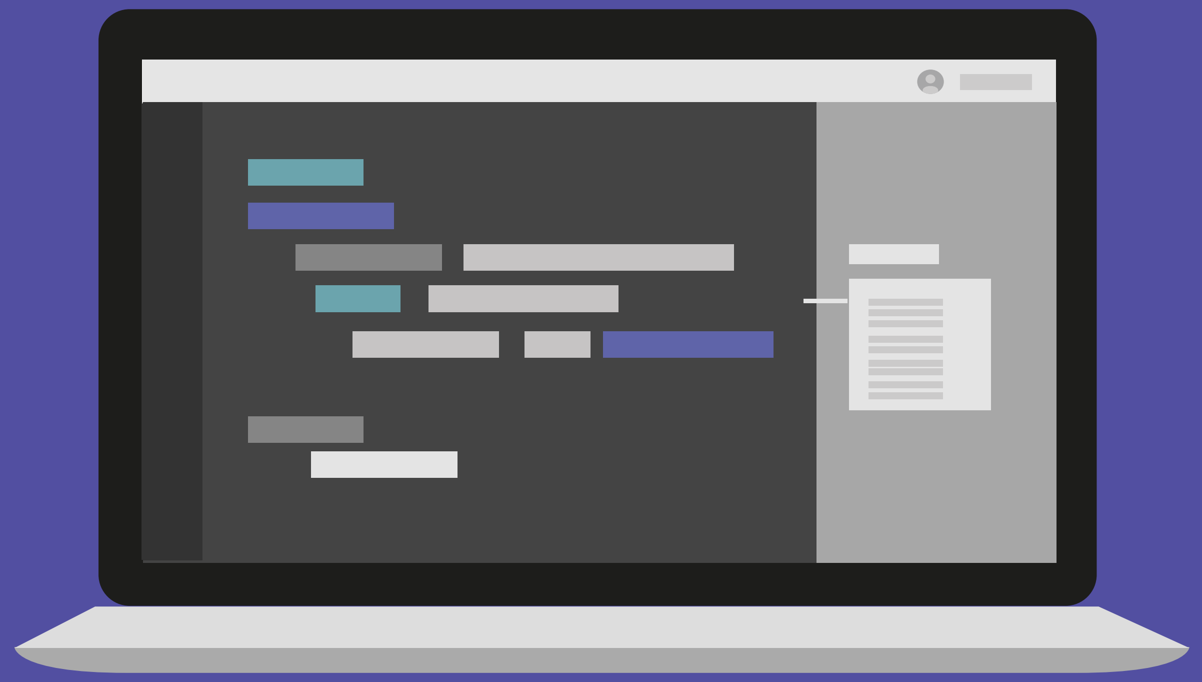
입력의 예

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

출력의 예

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

[실습 2] 퀵정렬 구현하기



재귀함수 디자인

[실습3] 올바른 괄호인지 판단하기

짝이 올바른 괄호라면 YES, 아니면 NO

입력의 예	출력의 예
(())()	YES
(())())()	NO

재귀함수 디자인

재귀함수를 디자인 하기 위한 세가지 단계

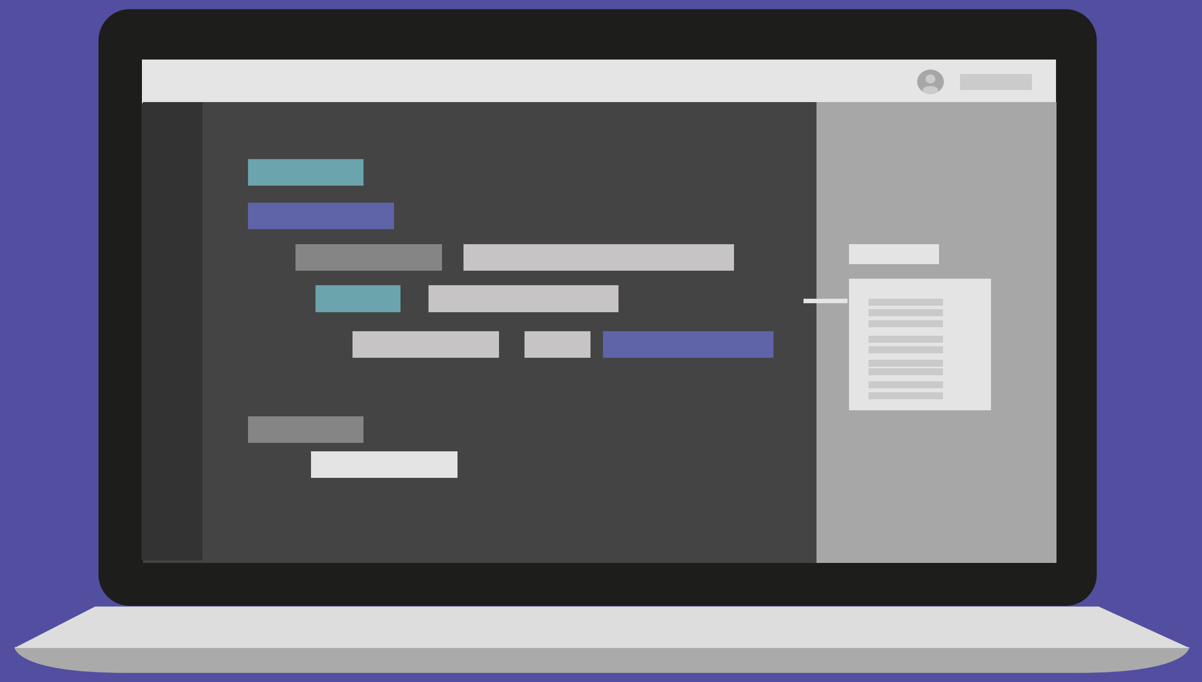
1. 함수의 **정의**를 명확히 한다.
2. **기저 조건**(Base condition)에서 함수가 제대로 동작하게 작성한다.
3. 함수가 **작은 input**에 대하여 제대로 동작한다고 가정하고 함수를 완성한다.

재귀함수 디자인

isRightParenthesis(p)

p가 올바른 괄호이면
“YES”, 아니면 “NO”를 반환하는 함수

[실습 3] 올바른 괄호인지 판단하기



요약

요약

알고리즘은 **“문제를 해결하는 방법”**이다.

재귀호출은 알고리즘에서 **매우 중요**하다.

/* elice */

문의 및 연락처

academy.elice.io

contact@elice.io

facebook.com/elice.io

medium.com/elice