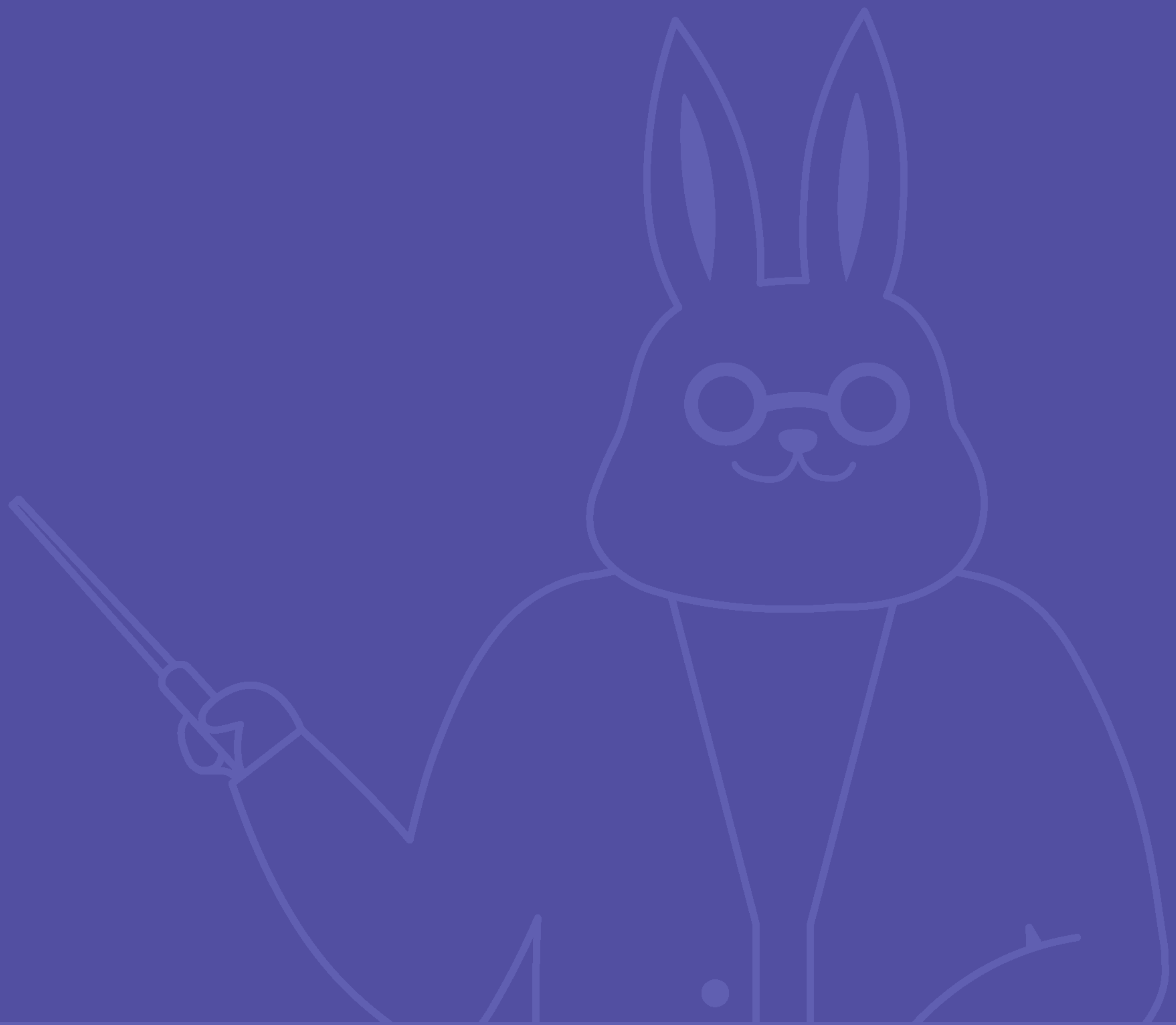
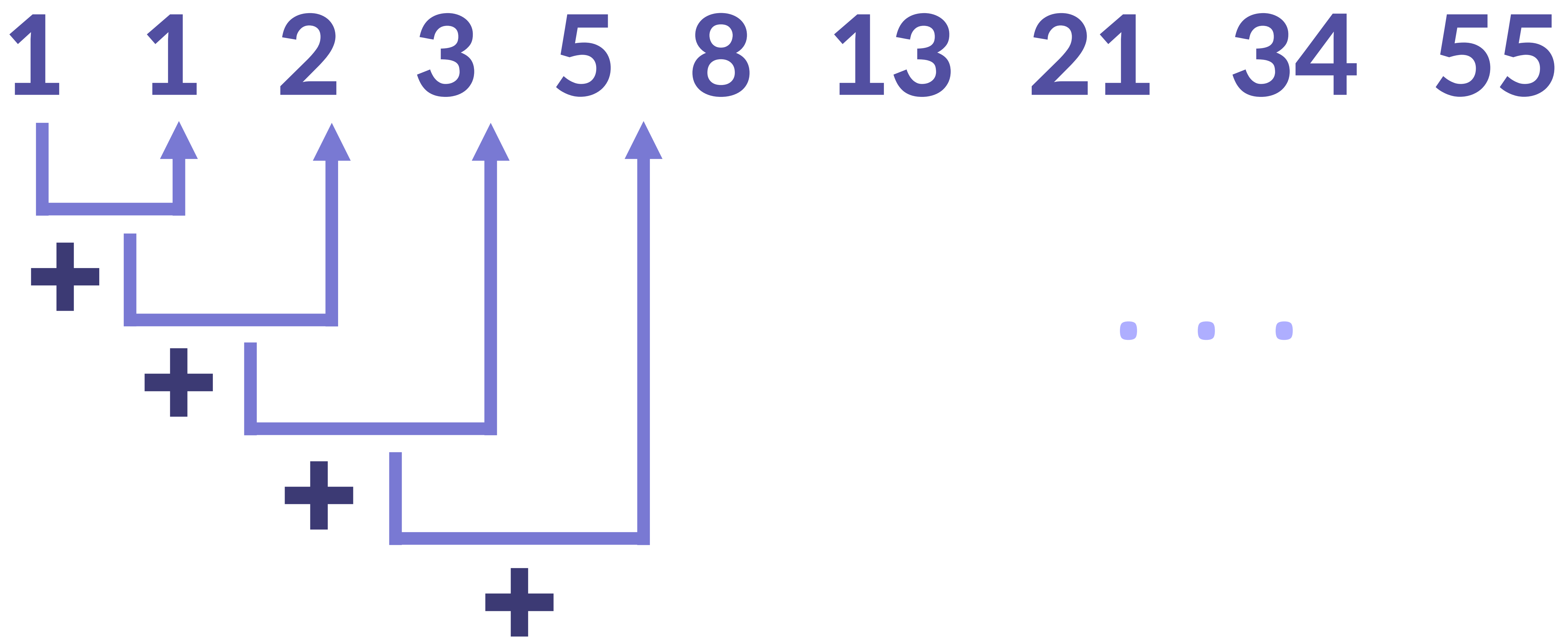
알고리즘의 정석 II

1장 동적계획법



Contents

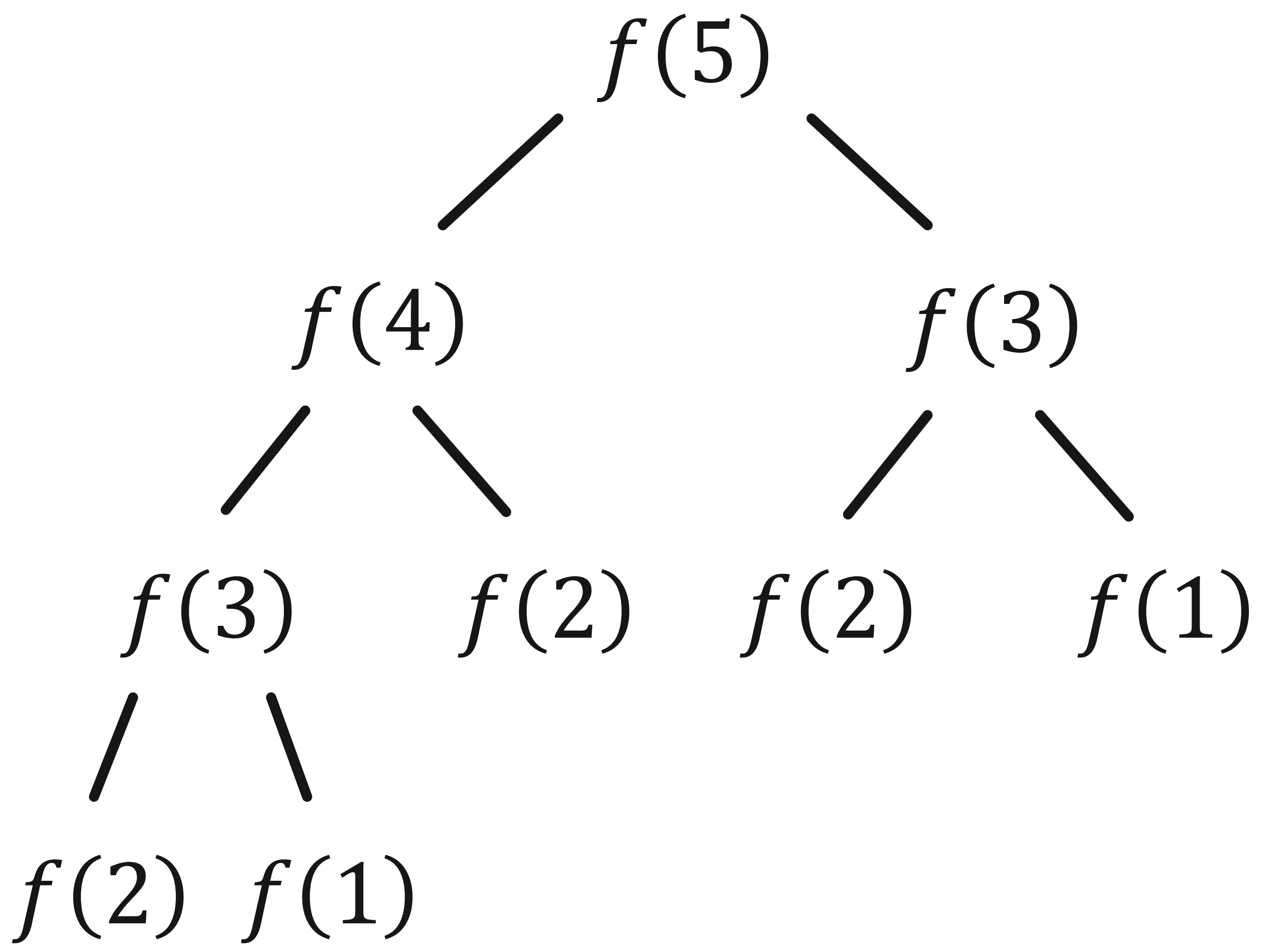
피보나치 수열이란?



재귀를 통한 피보나치 수열 만들기

Example

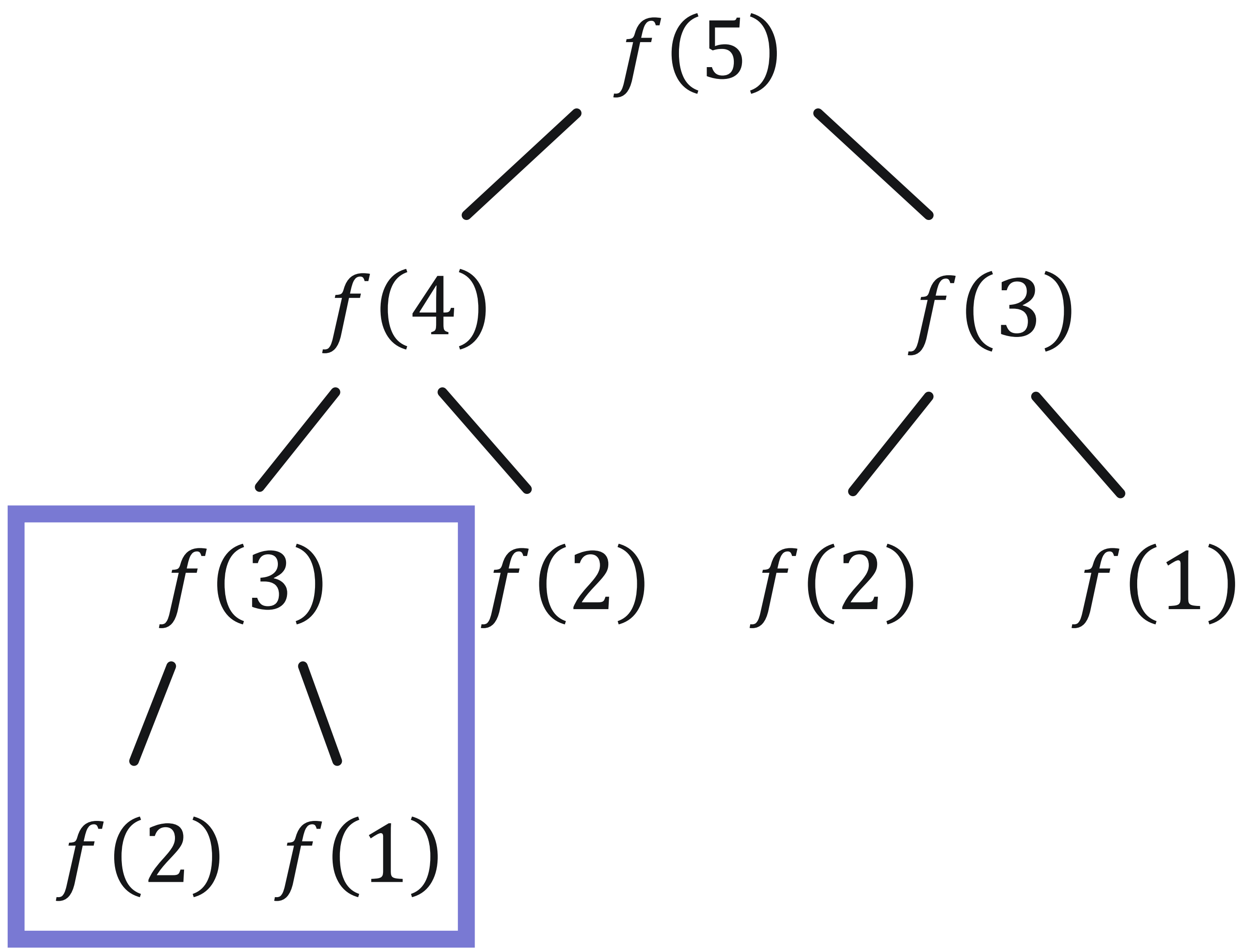
```python
def fibo(n):
    if n < 3:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```



재귀를 통한 피보나치 수열 만들기

Example

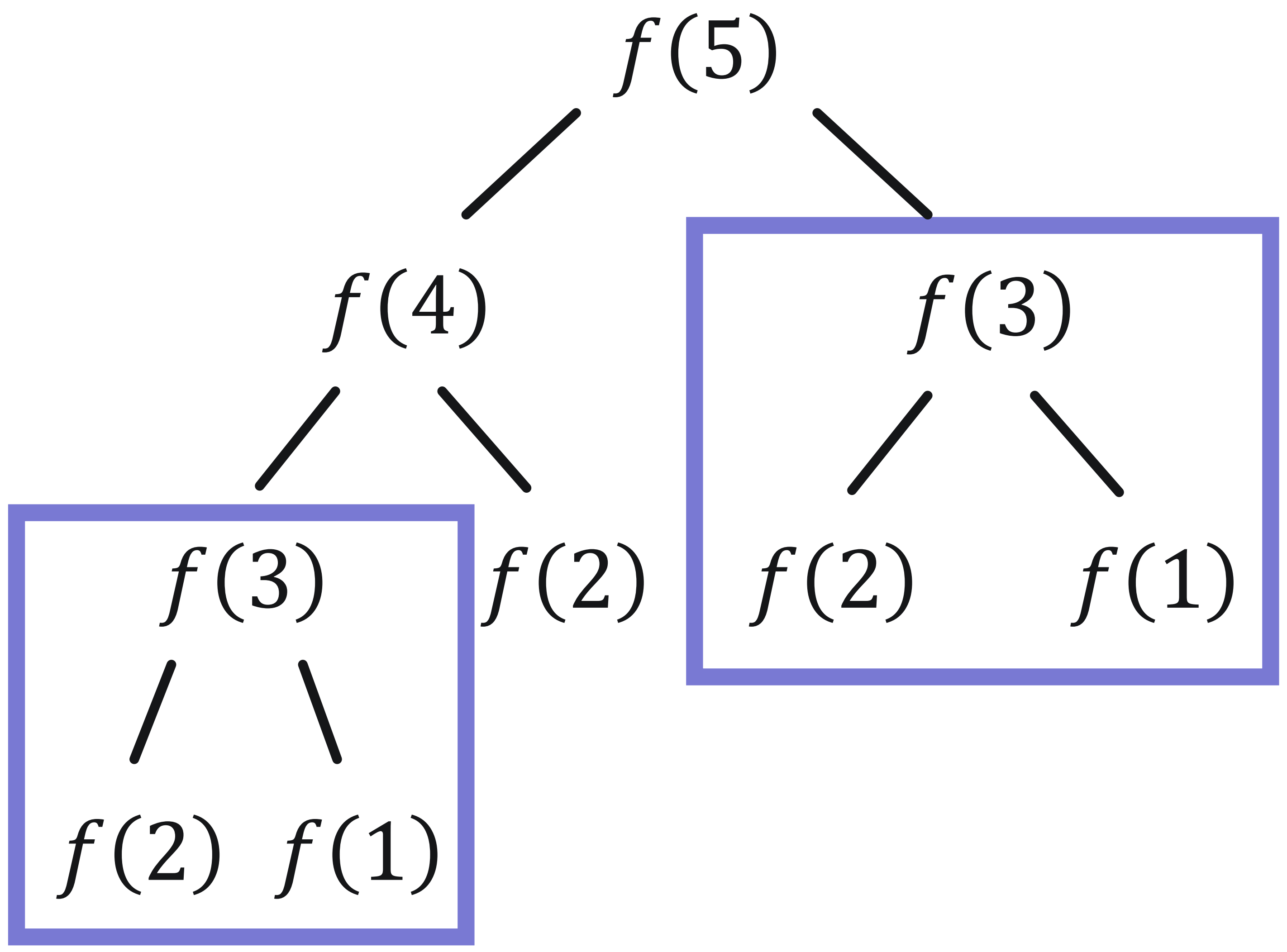
```python
def fibo(n):
    if n < 3:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```



재귀를 통한 피보나치 수열 만들기

Example

```python
def fibo(n):
    if n < 3:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```



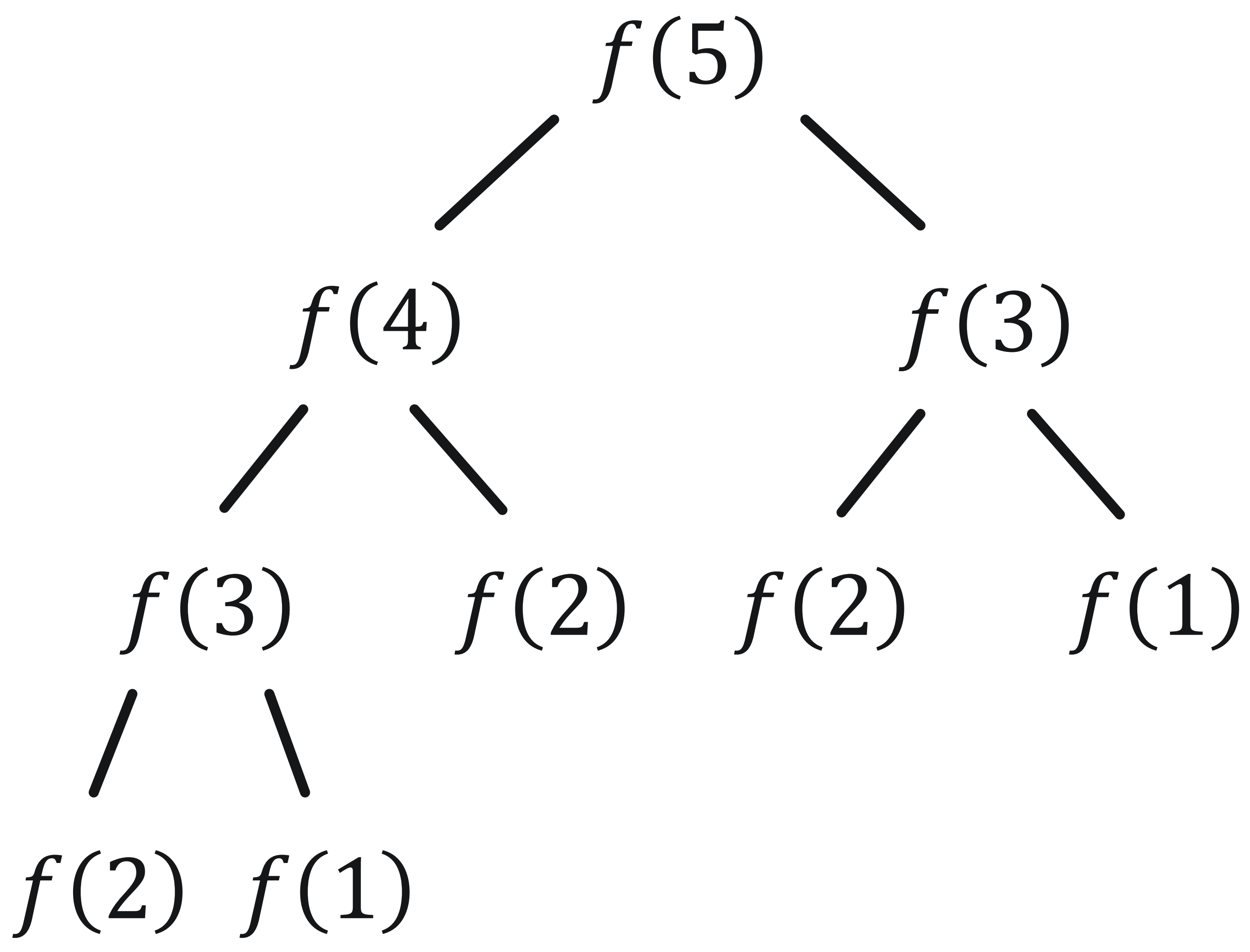
동적계획법이란?

복잡한 문제를 간단한 **여러 개의 하위 문제로 나누어** 푸는 방법

- 이 때, 하위 문제의 답을 저장하여 **중복 연산을 하지 않습니다.**

Example

```python
def fibo(n):
    if n < 3:
        return 1
    else:
        return fibo(n-1) + fibo(n-2)
```



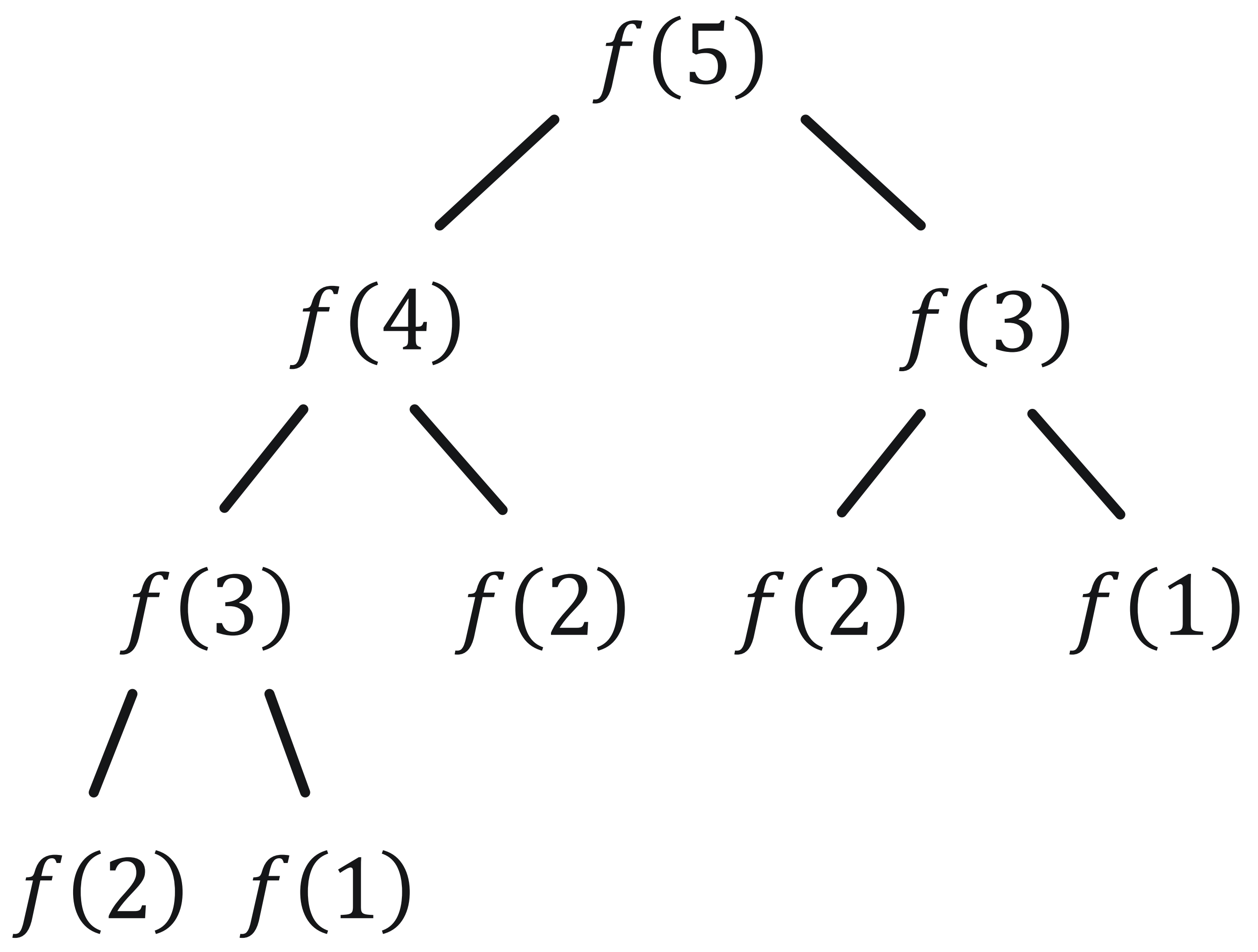
동적계획법이란?

복잡한 문제를 간단한 **여러 개의 하위 문제로 나누어** 푸는 방법

- 이 때, 하위 문제의 답을 저장하여 **중복 연산을 하지 않습니다.**

Example

```
fibonacci = {1: 1, 2: 1}

def fibo(n):
    if n in fibonacci:
        return fibonacci[n]
    else:
        fibonacci[n] = fibo(n-1) + fibo(n-2)
        return fibonacci[n]
```



동적계획법이란?

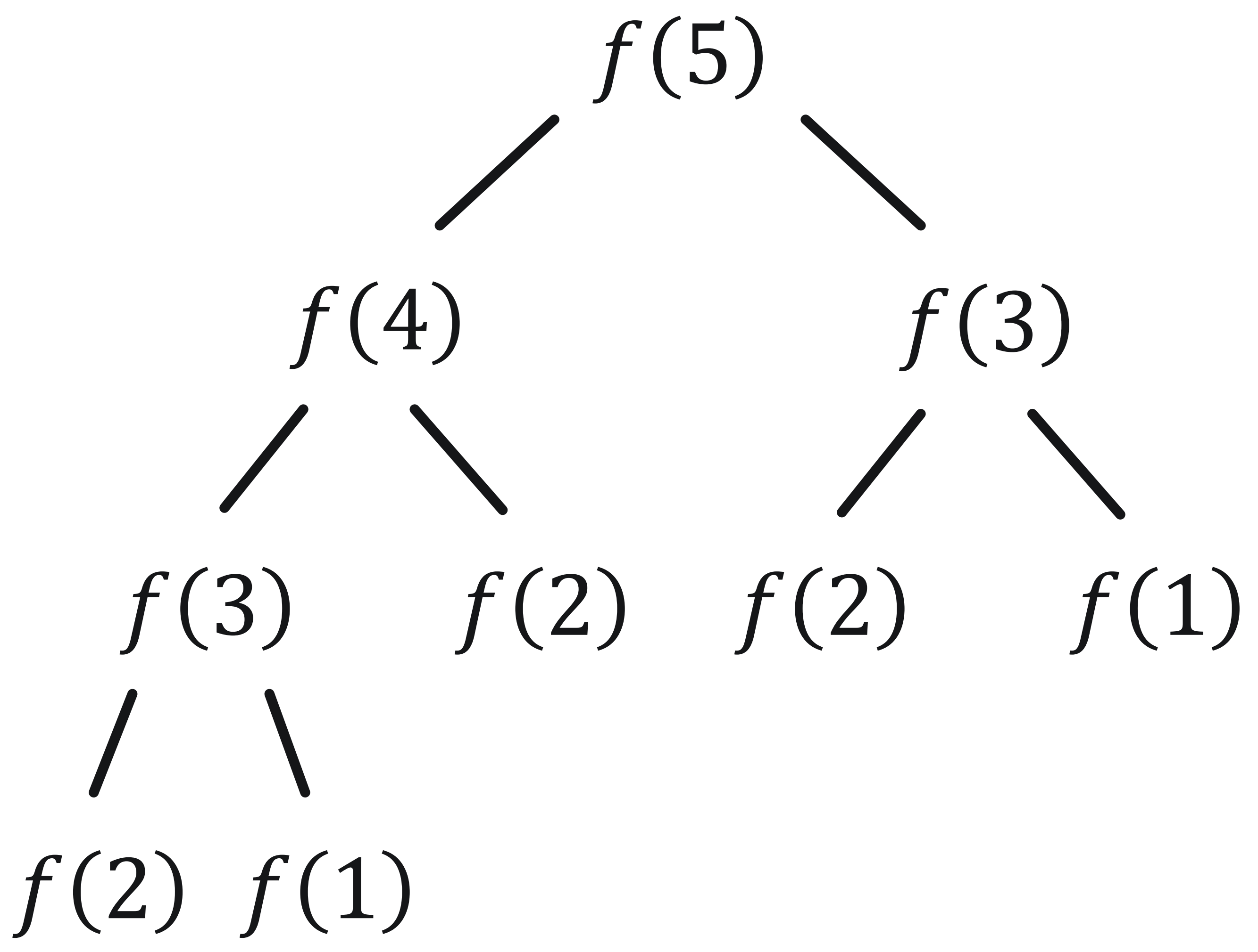
복잡한 문제를 간단한 **여러 개의 하위 문제로 나누어** 푸는 방법

- 이 때, 하위 문제의 답을 저장하여 **중복 연산을 하지 않습니다.**

Example

cache

```
fibonacci = {1: 1, 2: 1}

def fibo(n):
    if n in fibonacci:
        return fibonacci[n]
    else:
        fibonacci[n] = fibo(n-1) + fibo(n-2)
        return fibonacci[n]
```



동적계획법이란?

복잡한 문제를 간단한 **여러 개의 하위 문제로 나누어** 푸는 방법

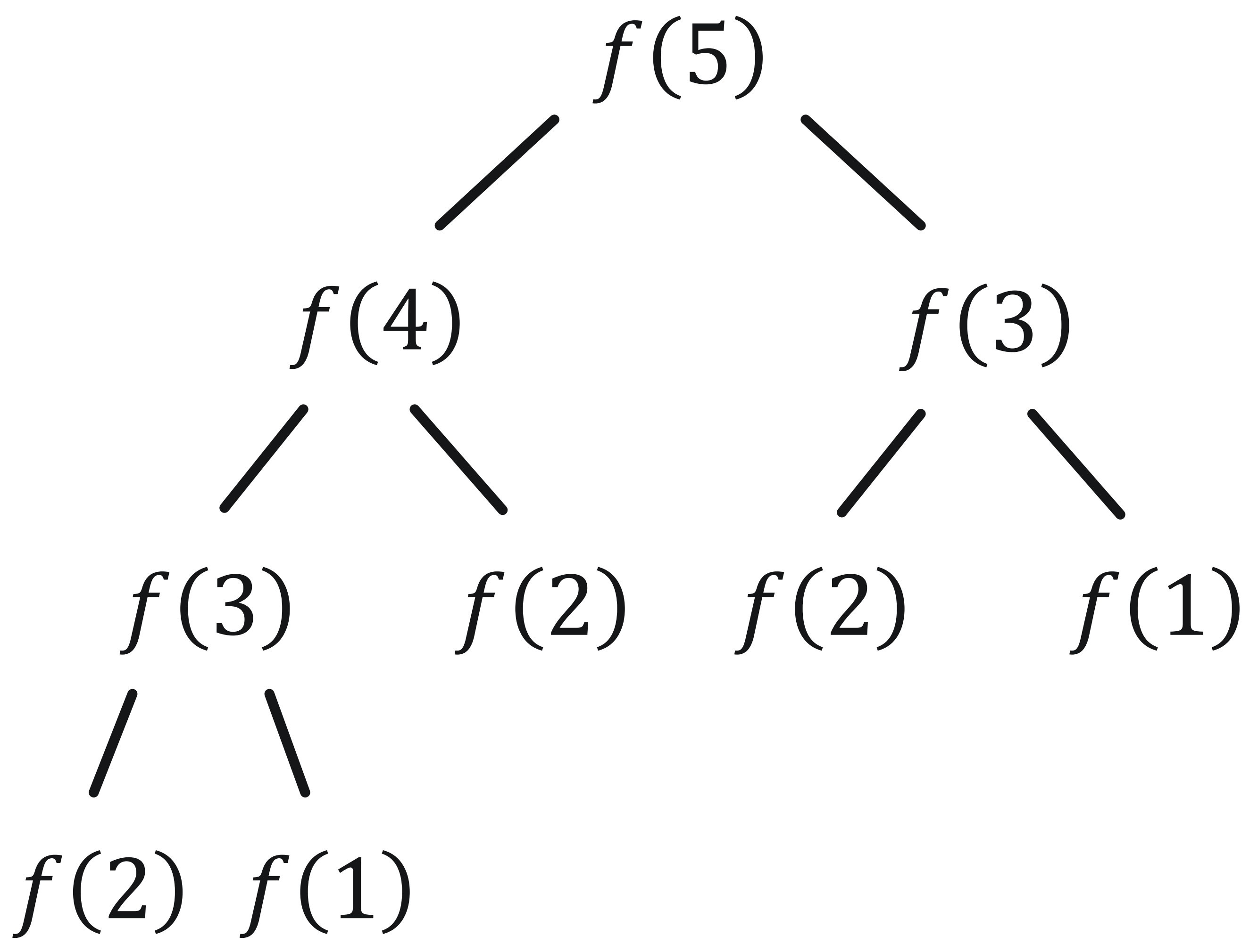
- 이 때, 하위 문제의 답을 저장하여 **중복 연산을 하지 않습니다.**

Example

cache

memoization

```
fibonacci = {1: 1, 2: 1}

def fibo(n):
    if n in fibonacci:
        return fibonacci[n]
    else:
        fibonacci[n] = fibo(n-1) + fibo(n-2)
        return fibonacci[n]
```



동적계획법이란?

복잡한 문제를 간단한 **여러 개의 하위 문제로 나누어** 푸는 방법

- 이 때, 하위 문제의 답을 저장하여 **중복 연산을 하지 않습니다.**

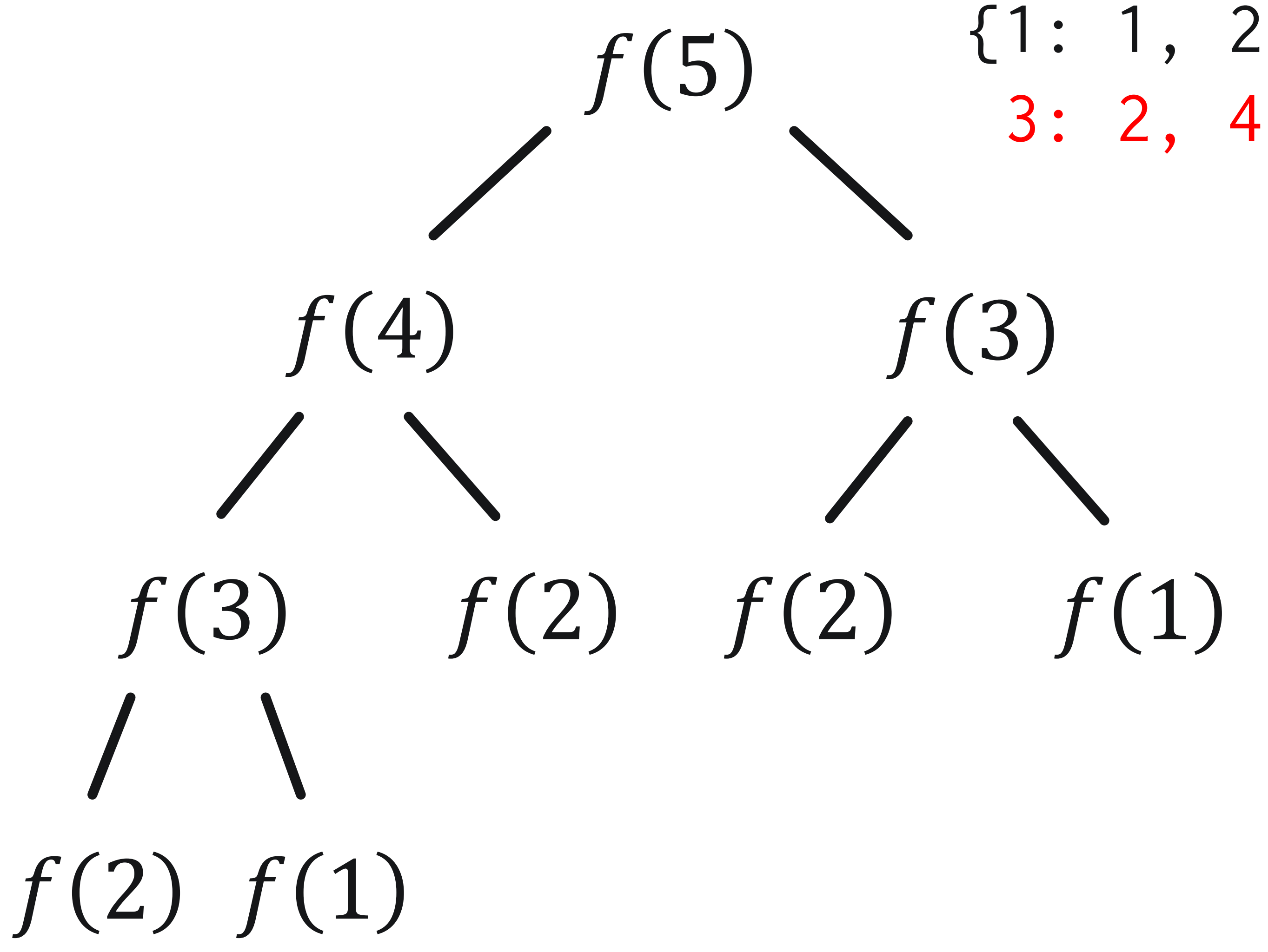
Example

cache

memoization

```python
fibonacci = {1: 1, 2: 1}

def fibo(n):
    if n in fibonacci:
        return fibonacci[n]
    else:
        fibonacci[n] = fibo(n-1) + fibo(n-2)
        return fibonacci[n]
```

```
fibonacci =
    {1: 1, 2: 1,
     3: 2, 4: 3}
```



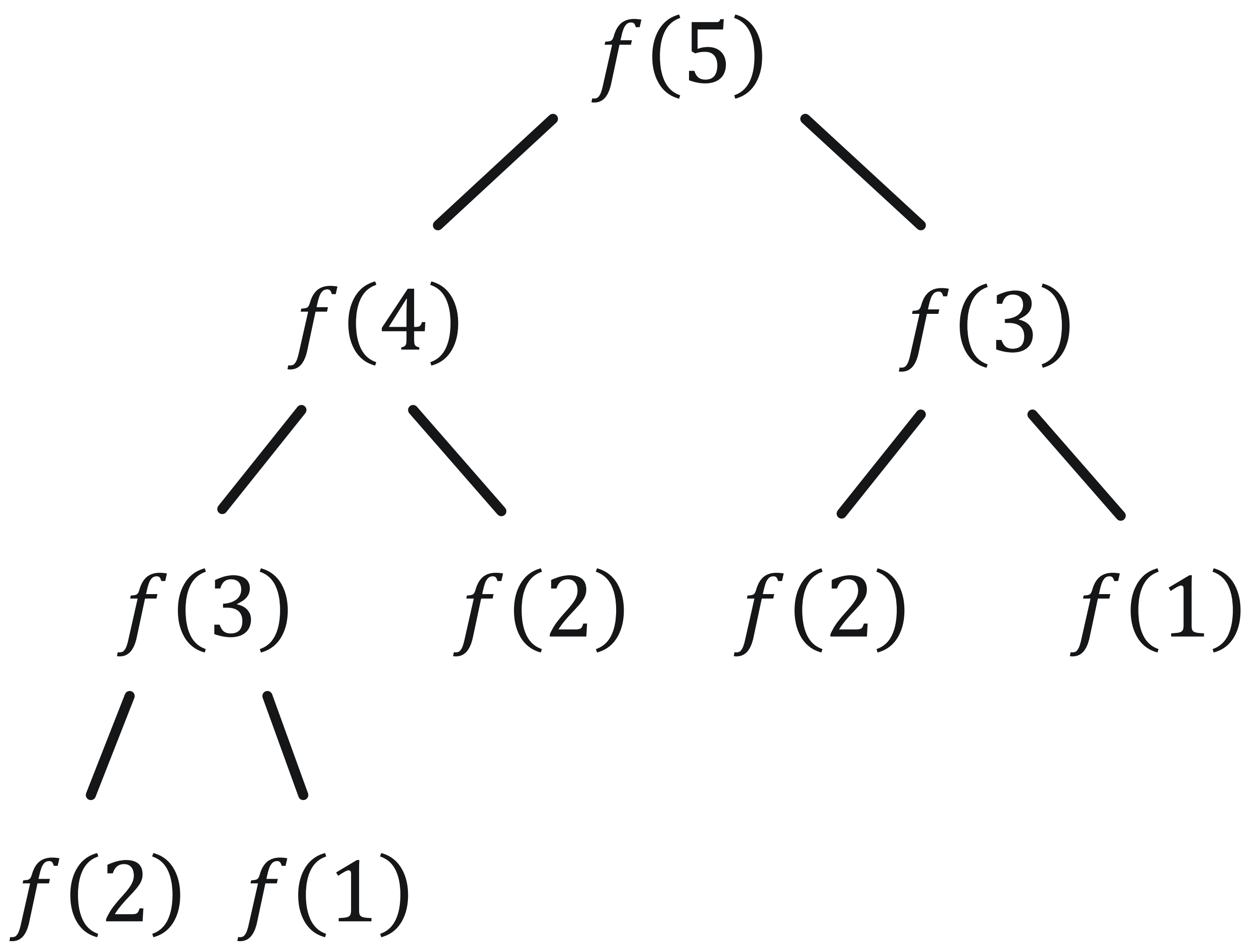
동적계획법 문제의 특성

중복되는 부분문제 (overlapping subproblems)

작은 하위문제들이 중복되어 나타난다.

최적 부분 구조 (optimal substructure)

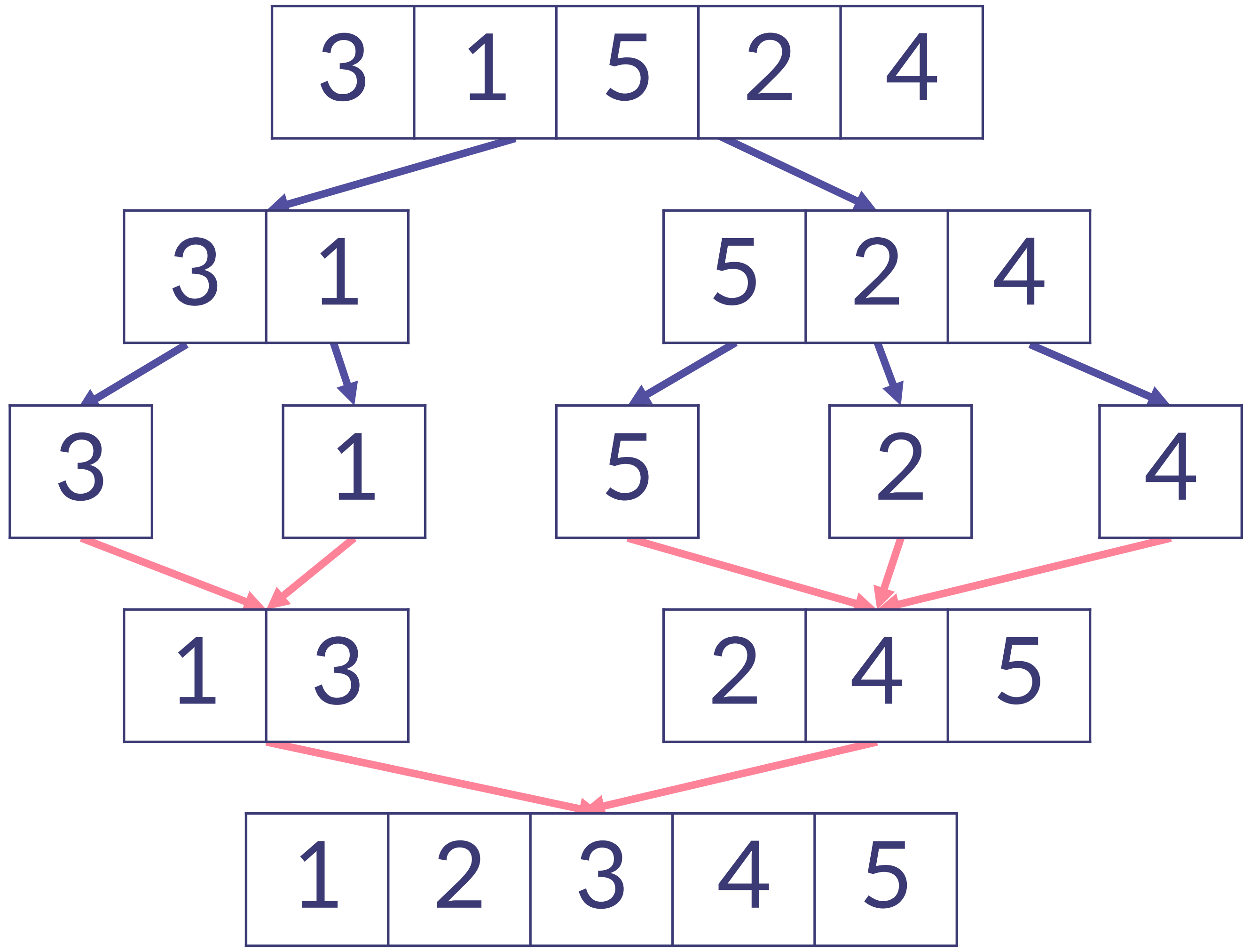
최적해는 부분 문제의 최적해로부터 구할 수 있다.



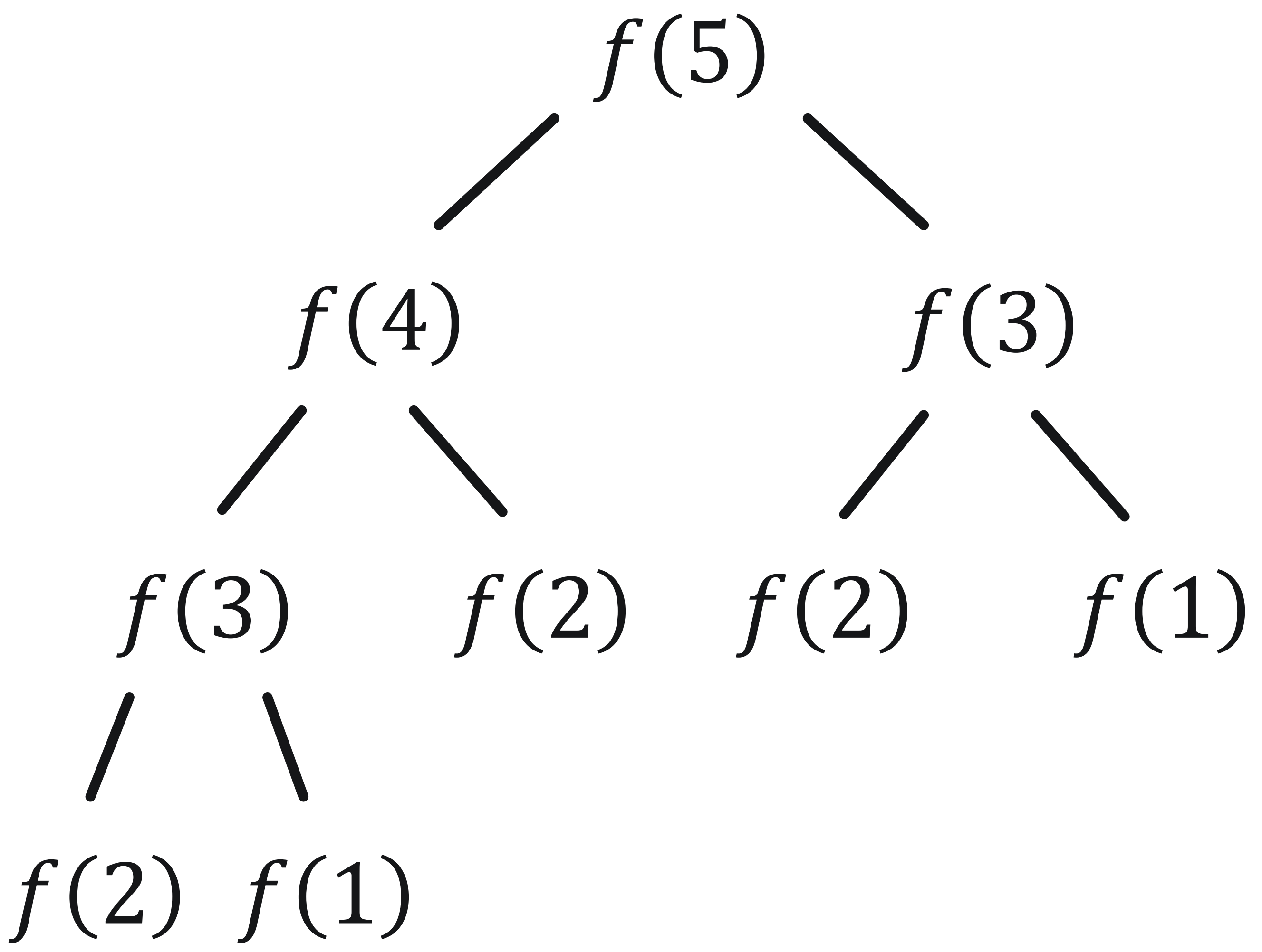
분할정복법과 동적계획법의 차이

중복되는 부분문제 (overlapping subproblems) 작은 하위문제들이 중복되어 나타난다.

- 분할정복법



- 동적계획법



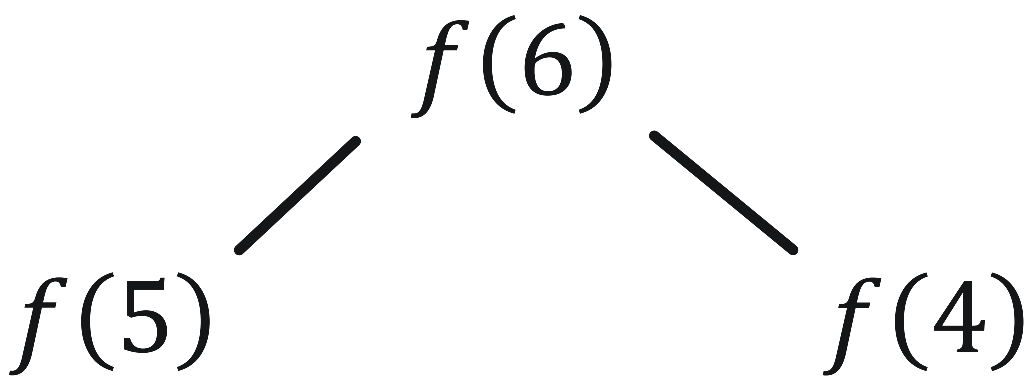
시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출
- 동적계획법

$$f(6)$$

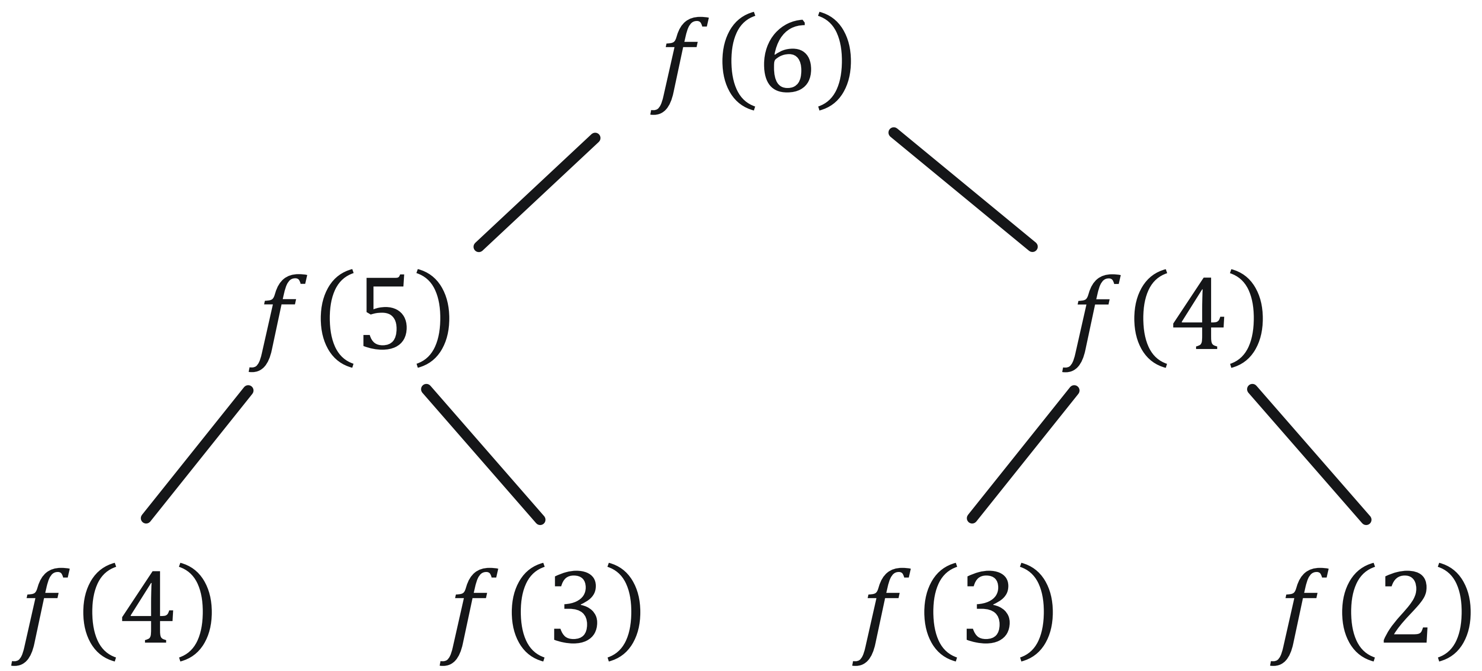
시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출
- 동적계획법



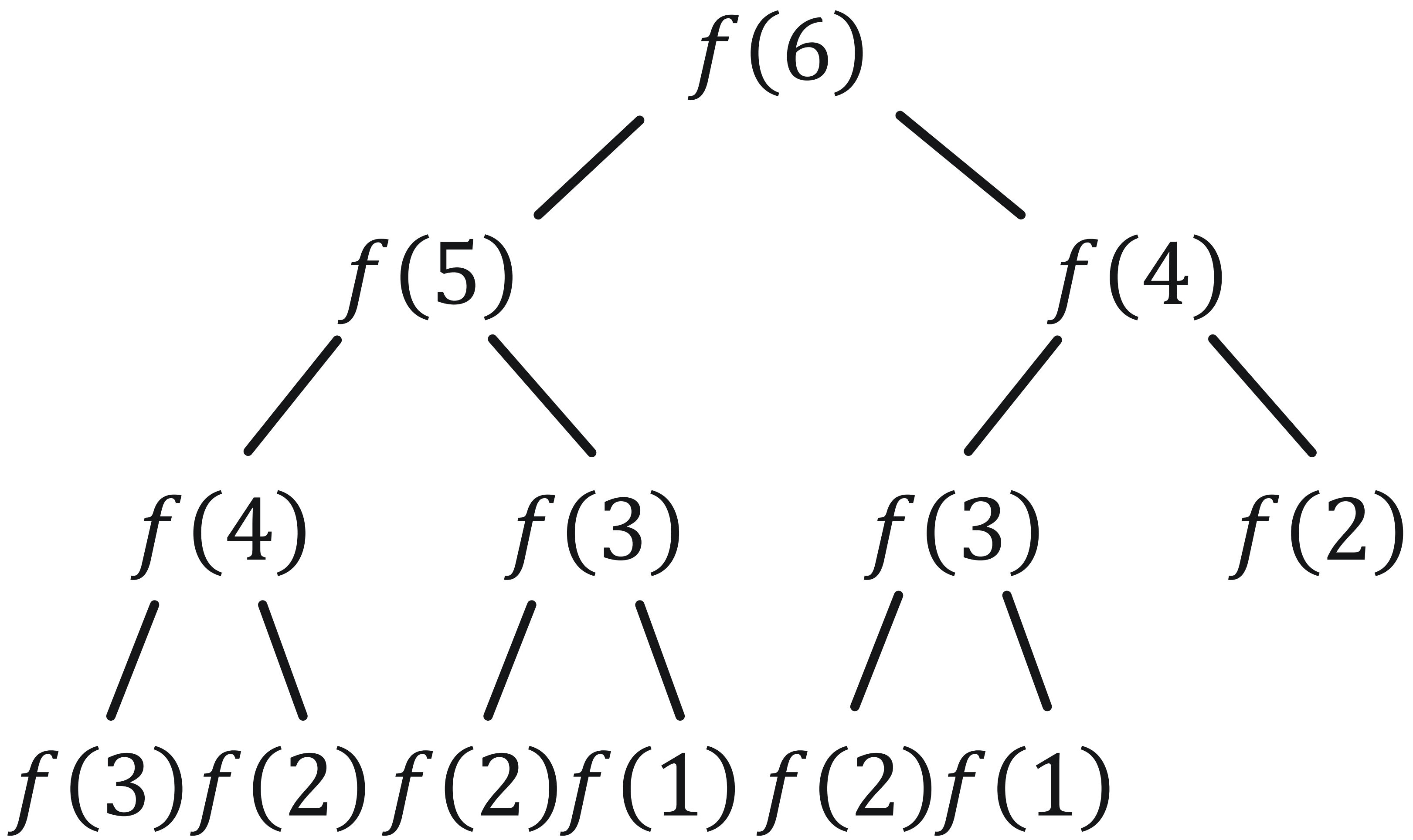
시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출
- 동적계획법



시간복잡도 - 시간이 얼마나 절약이 될까?

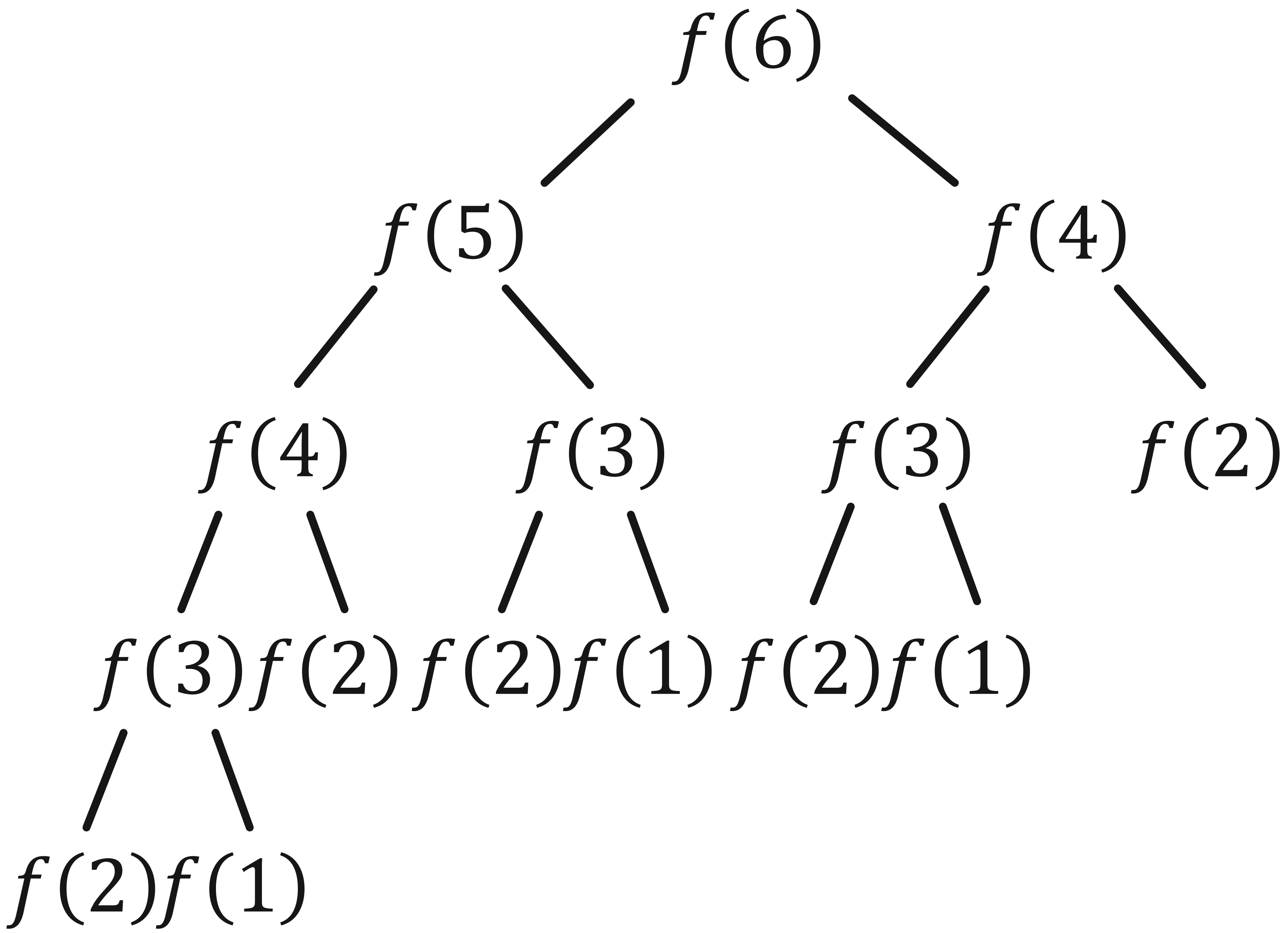
- 간단한 재귀호출



- 동적계획법

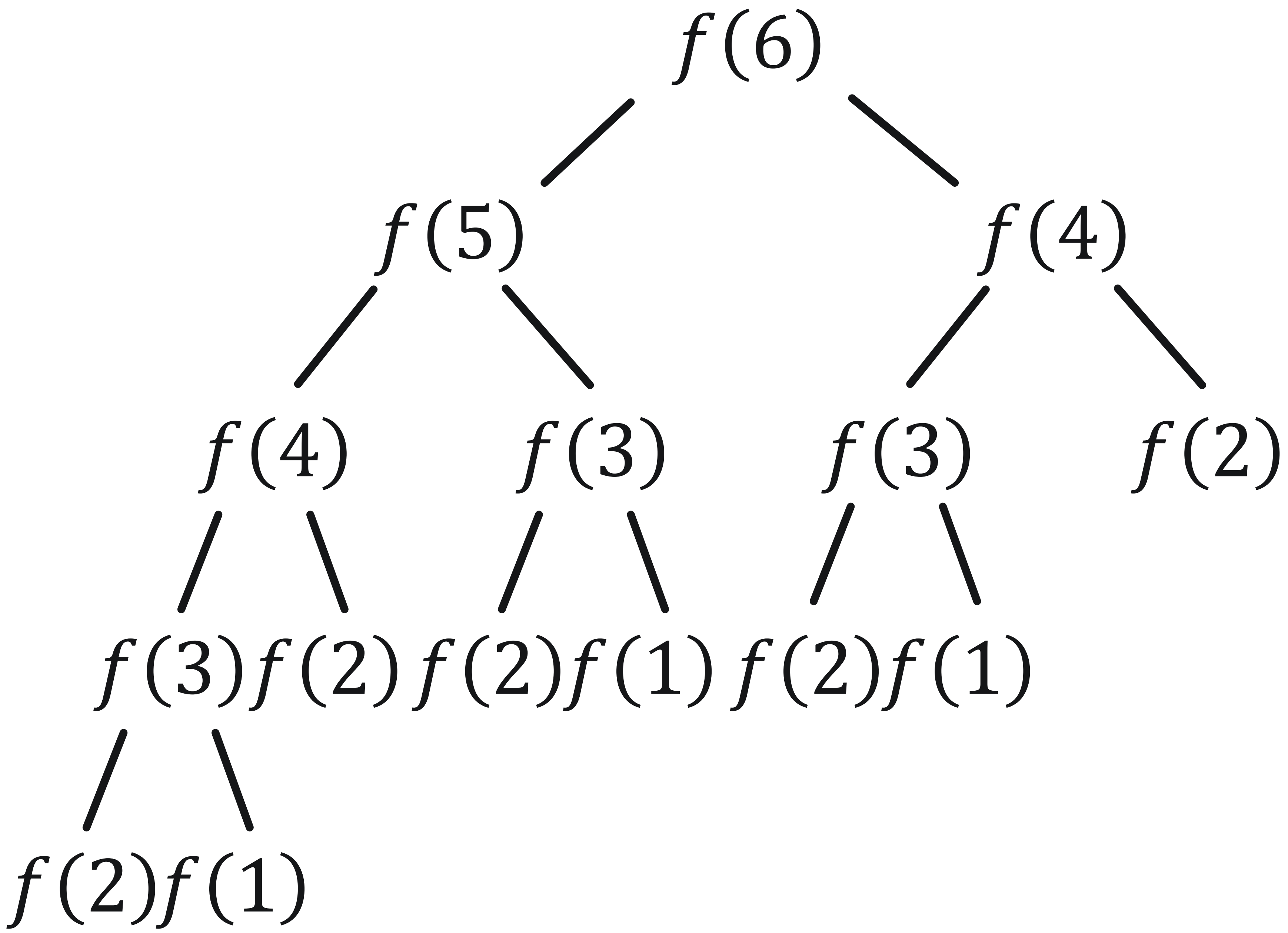
시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출
- 동적계획법

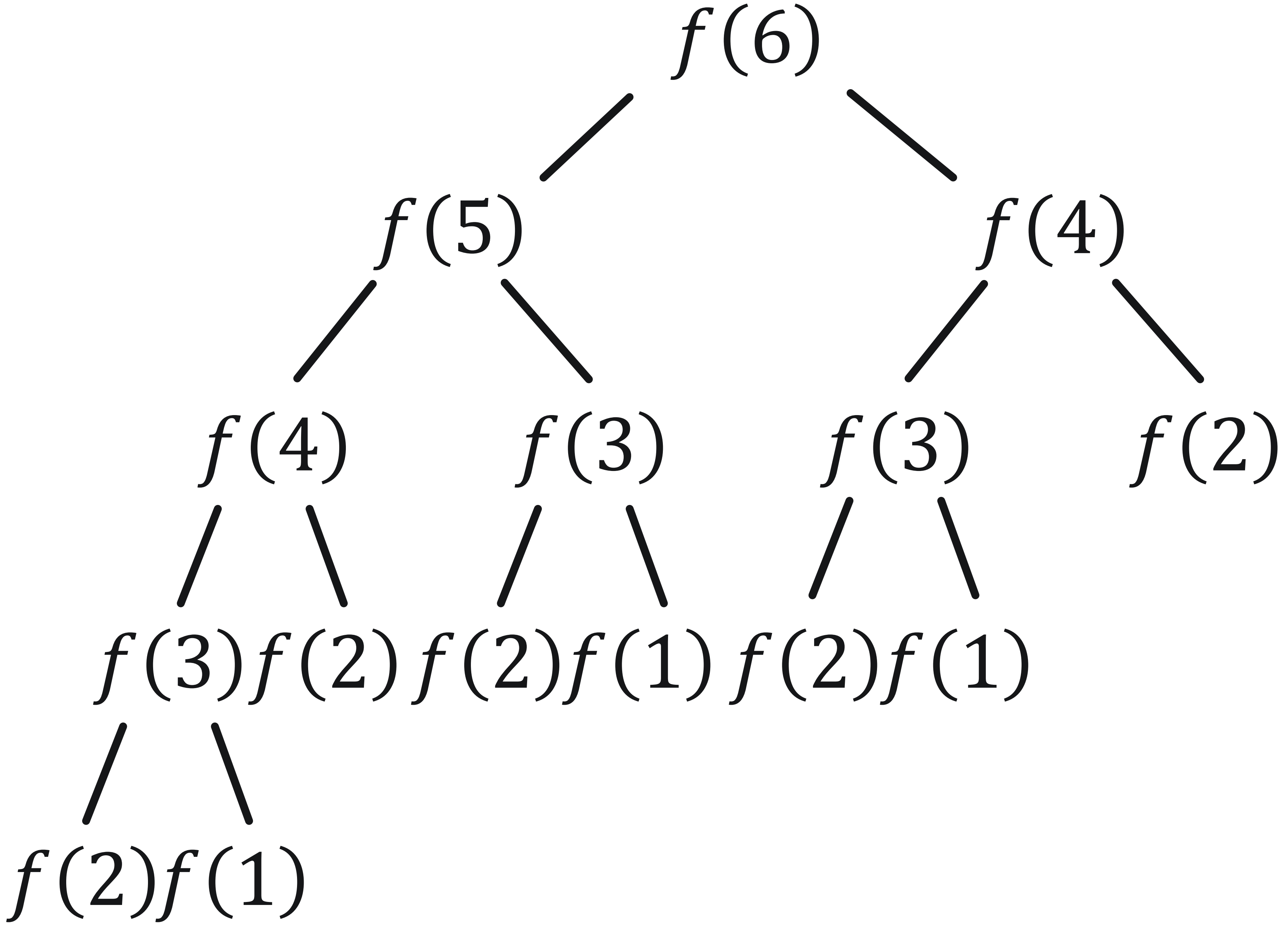


시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출

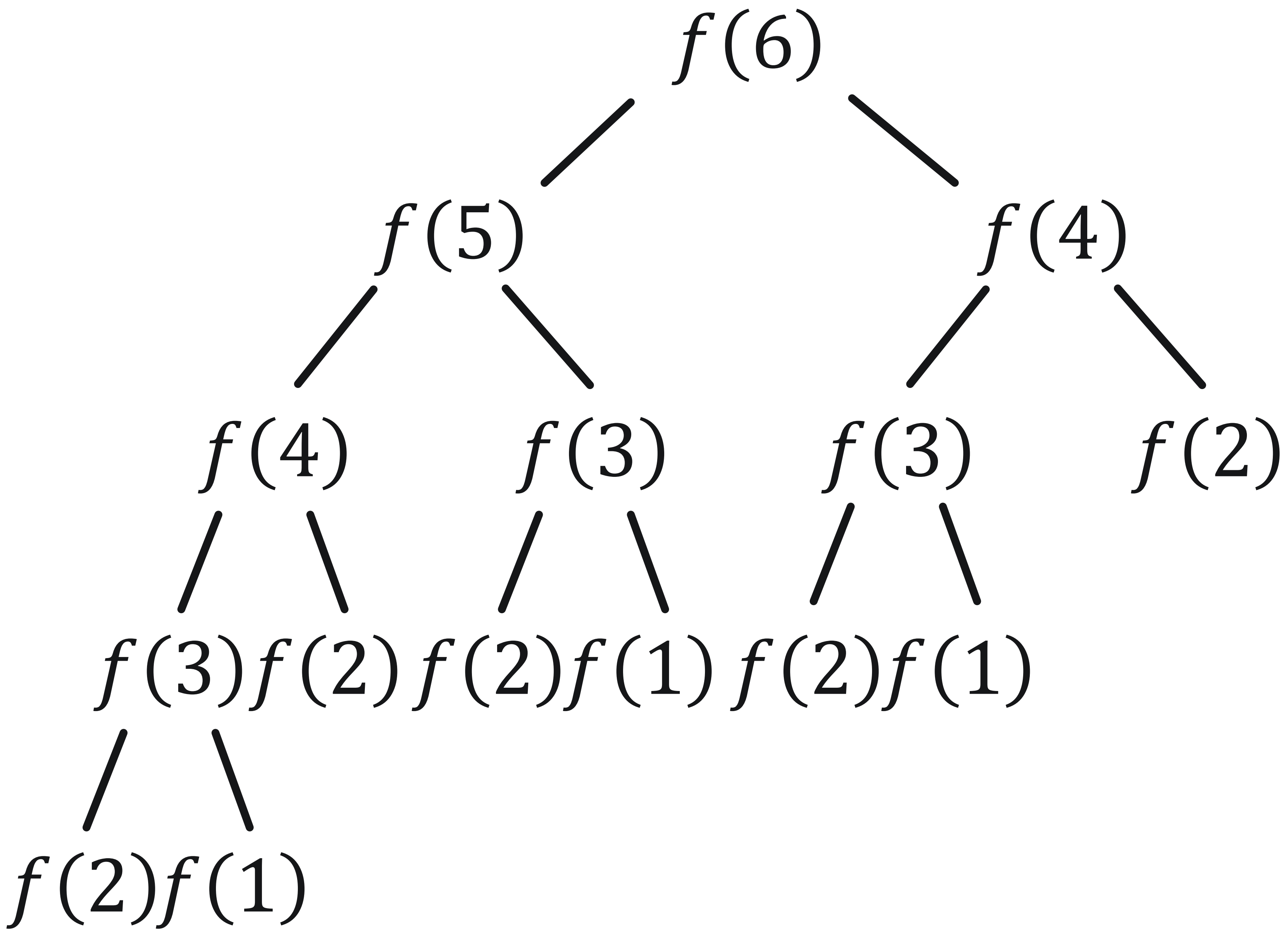


- 동적계획법

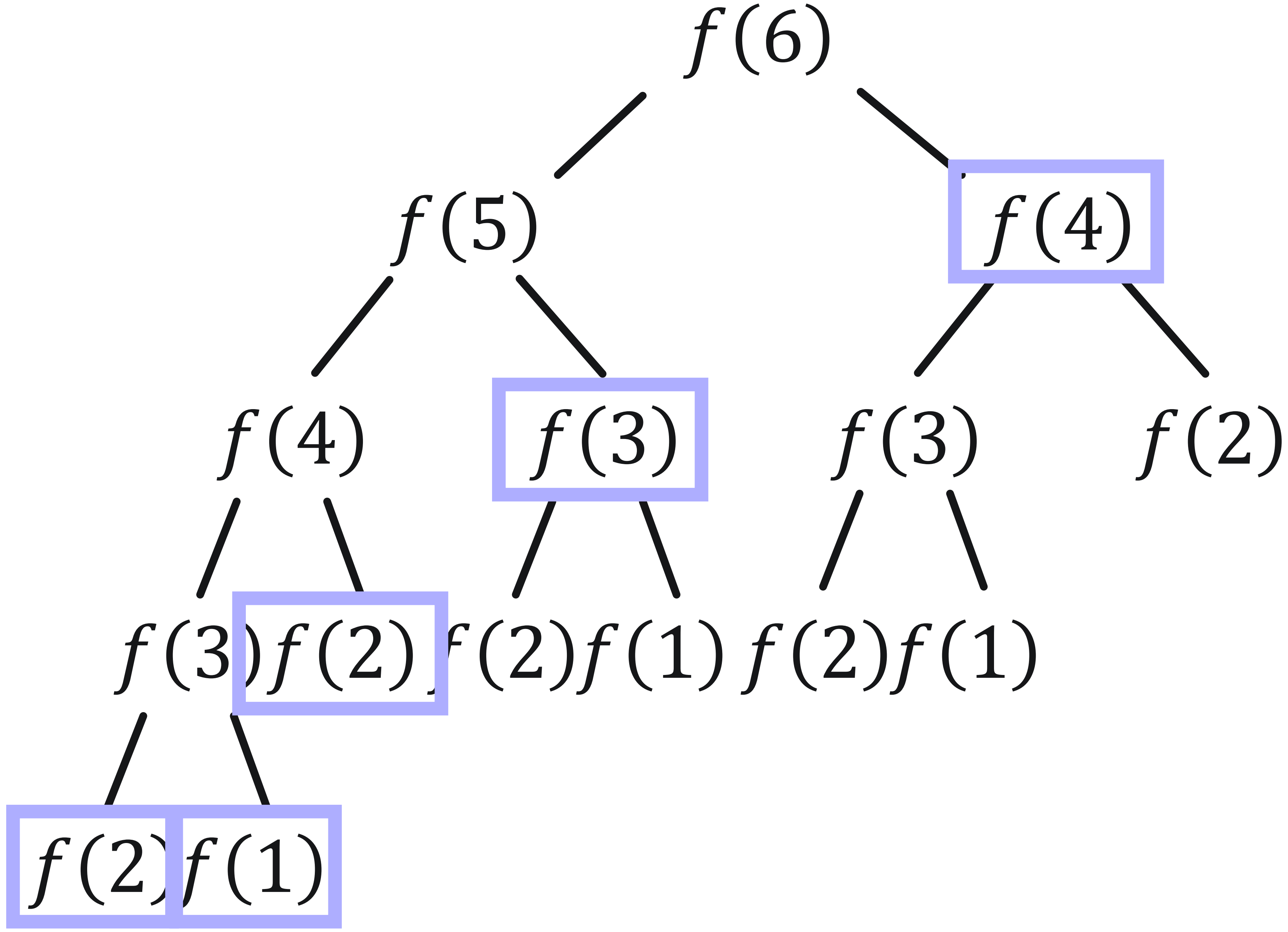


시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출

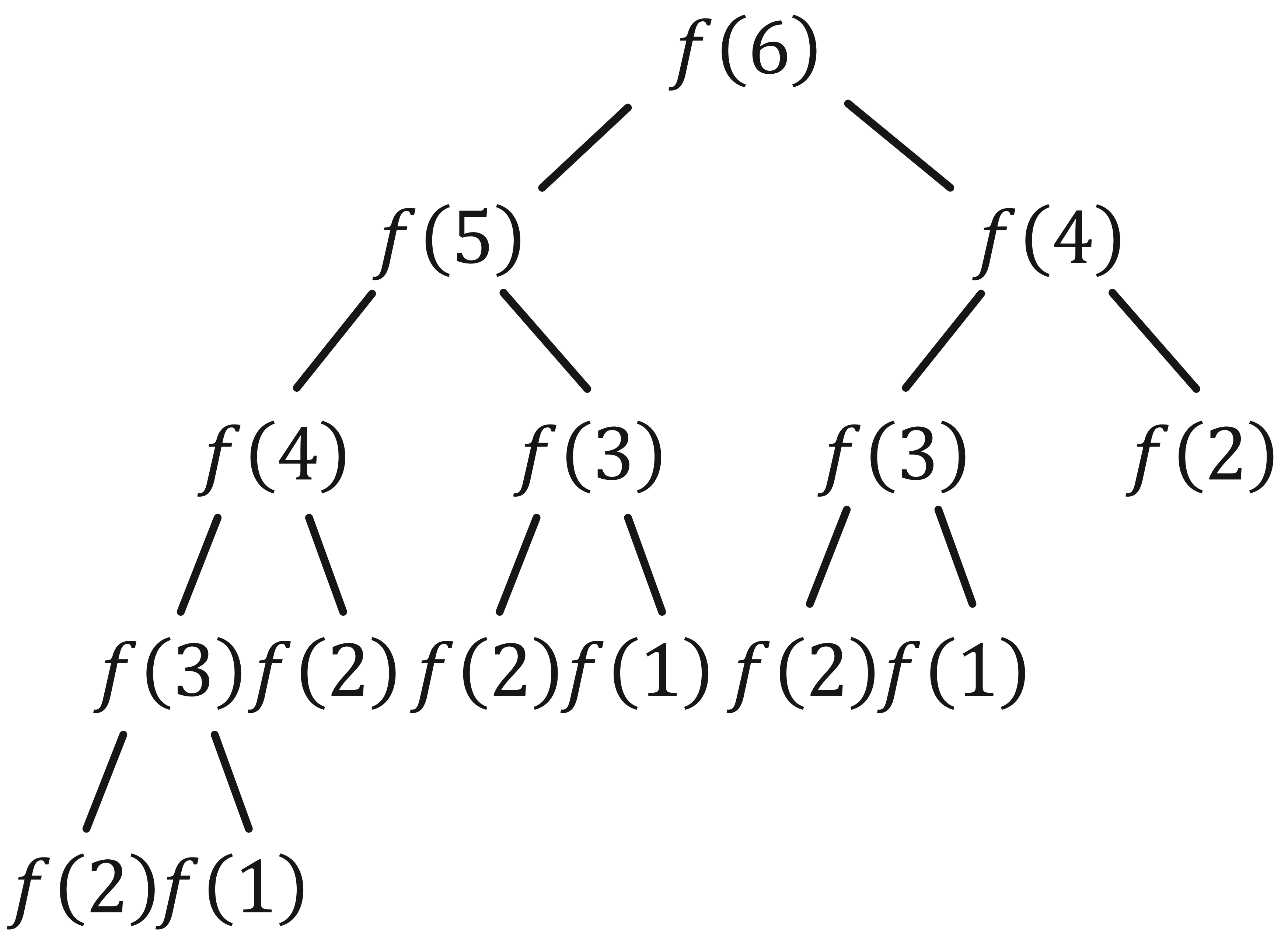


- 동적계획법

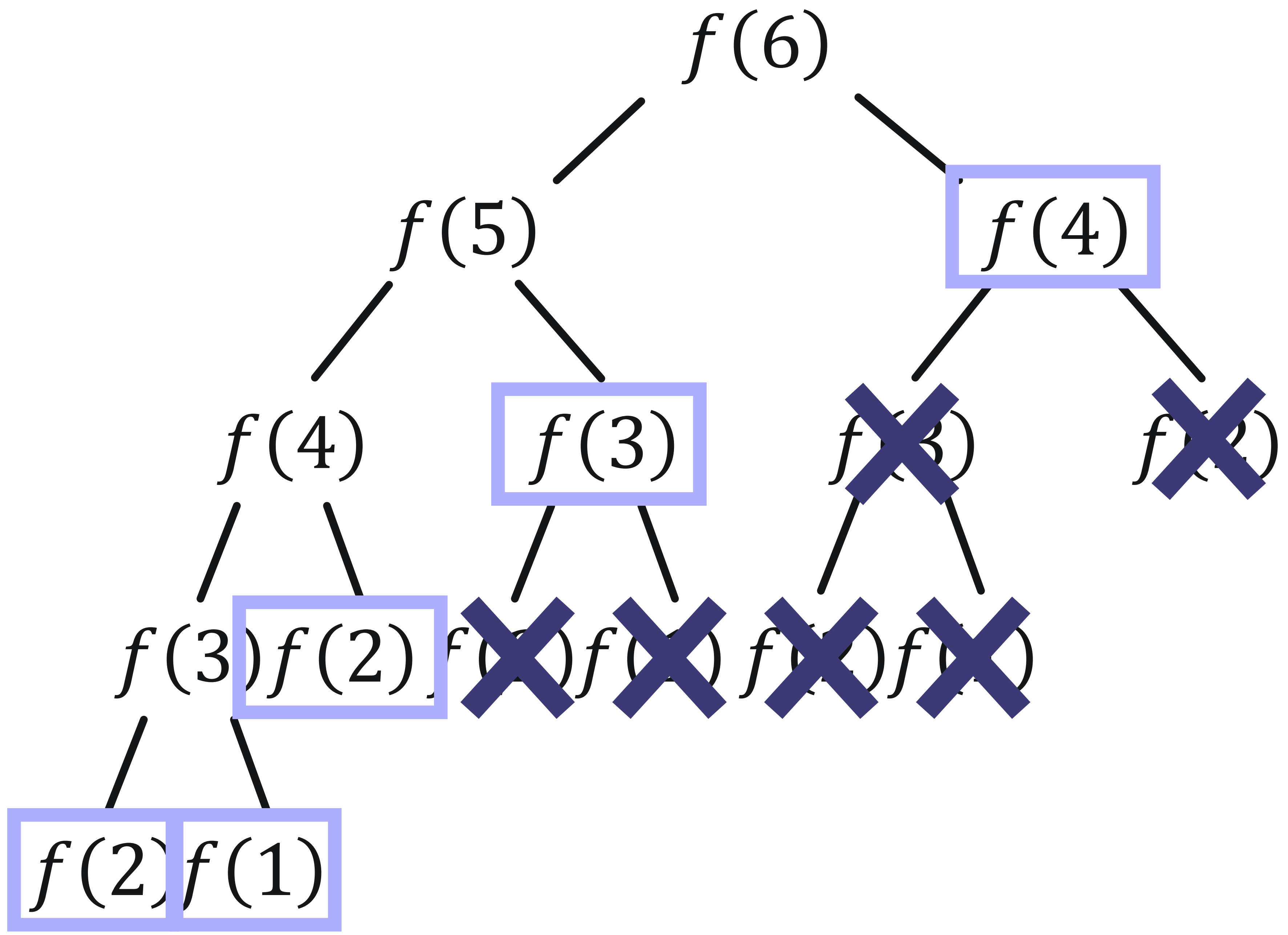


시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출

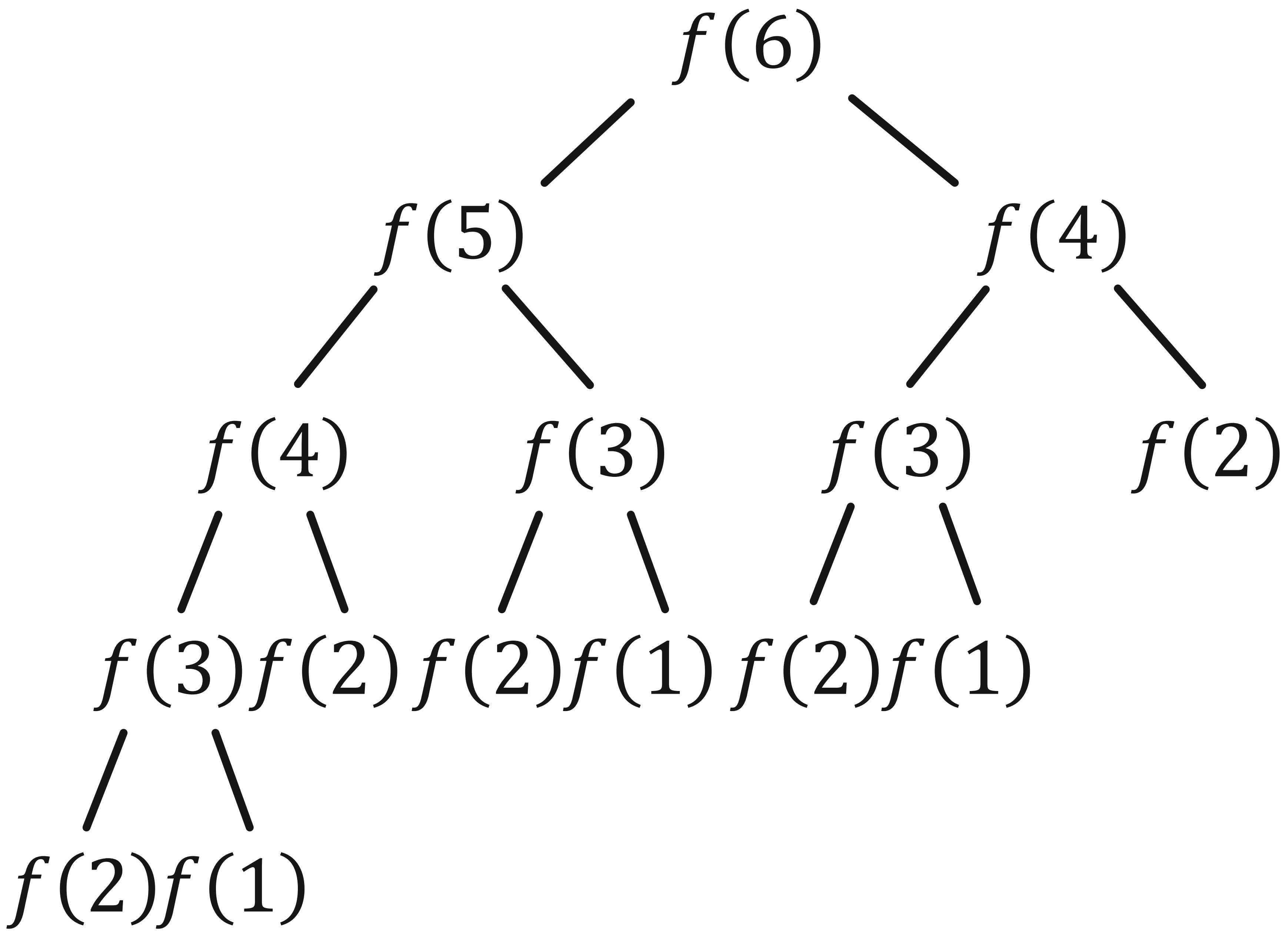


- 동적계획법

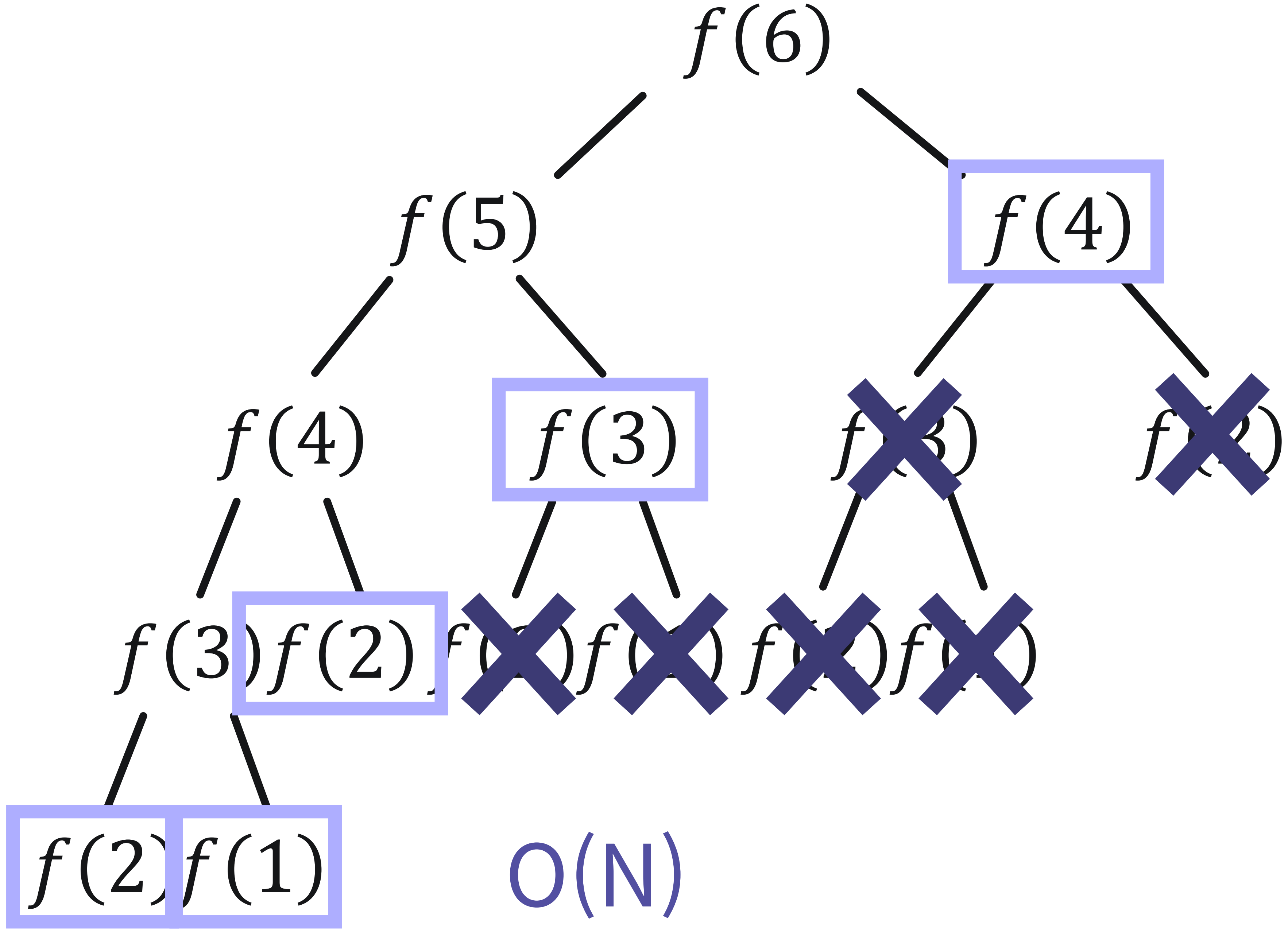


시간복잡도 - 시간이 얼마나 절약이 될까?

- 간단한 재귀호출



- 동적계획법



시간복잡도 - 시간이 얼마나 절약이 될까?

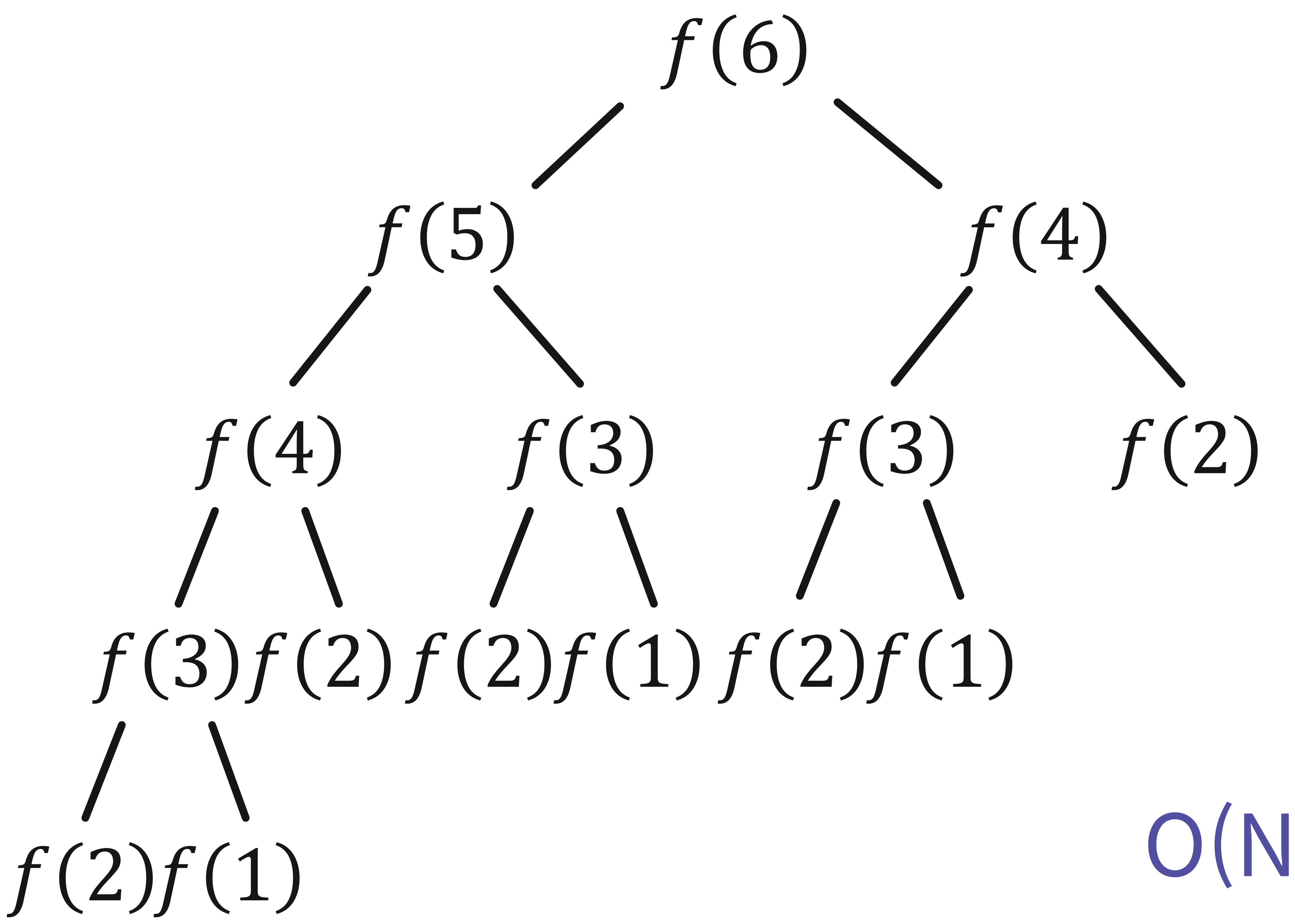
- 간단한 재귀호출

```
In [1]: def fibo_1(n):
            if n < 3:
                return 1
            else:
                return fibo_1(n-1)+ fibo_1(n-2)

In [2]: %%time
        fibo_1(30)

        Wall time: 306 ms

Out[2]: 832040
```

- 동적계획법

```
In [3]: fibonacci = {1: 1, 2: 1}

        def fibo_2(n):
            if n in fibonacci:
                return fibonacci[n]
            else:
                fibonacci[n] = fibo_2(n-1)+ fibo_2(n-2)
                return fibonacci[n]

In [4]: %%time
        fibo_2(30)

        Wall time: 0 ns

Out[4]: 832040
```

공간복잡도 - 공간을 얼마나 사용할까?



O(N)

동적계획법은
하위 문제들의 답을 저장해놓기 때문에,
하위 문제의 수만큼 저장공간이 필요하다

동적계획법을 구현하는 테크닉

점화식

복잡한 문제를 작은 하위문제로 표현한 식
예) $f(n) = f(n-1)+f(n-2)$

1. **구하고자 하는 값**이 무엇인지 정의한다
   $f(n)$: n번째 피보나치 수열

2. 구하고자 하는 값을 **부분문제들로 표현**한다
   피보나치 수열의 n번째 항은 n-1항과 n-2항의 합이다.
   $f(n) = f(n-1)+f(n-2)$

동적계획법을 구현하는 테크닉

점화식

복잡한 문제를 작은 하위문제로 표현한 식

예) f(n) = f(n-1)+f(n-2)

- **Top-down**
  1. 큰 문제를 작은 문제로 나눈다
  2. 작은문제를 풀어 return해준다

  재귀호출 식 방법

- **Bottom-up**
  1. 작은 문제부터 차례로 풀어 적는다.
  2. 크기를 조금씩 늘려서 문제를 푼다

  반복문 식 방법

동적계획법을 구현하는 두가지 테크닉

- Top-down

Example

```
fibonacci = {1: 1, 2: 1}

def fibo(n):
    if n in fibonacci:
        return fibonacci[n]
    else:
        fibonacci[n] = fibo(n-1) + fibo(n-2)
        return fibonacci[n]
```

- Bottom-up

Example

```
def fibo(n):
    fibonacci = [-1, 1, 1]
    if n < 3: return 1
    for i in range(3, n+1):
        fibonacci.append(
          fibonacci[i-1] + fibonacci[i-2])
    return fibonacci[n]
```

동적계획법 문제풀이 방법

1. 구하고자 하는 값 정의하기

2. 부분문제로 표현하여 점화식 구하기

3. 코드로 옮기기

동적계획법 문제풀이 방법

1. **구하고자 하는 값** 정의하기
   구하고자 하는 값이 무엇인지 정의한다

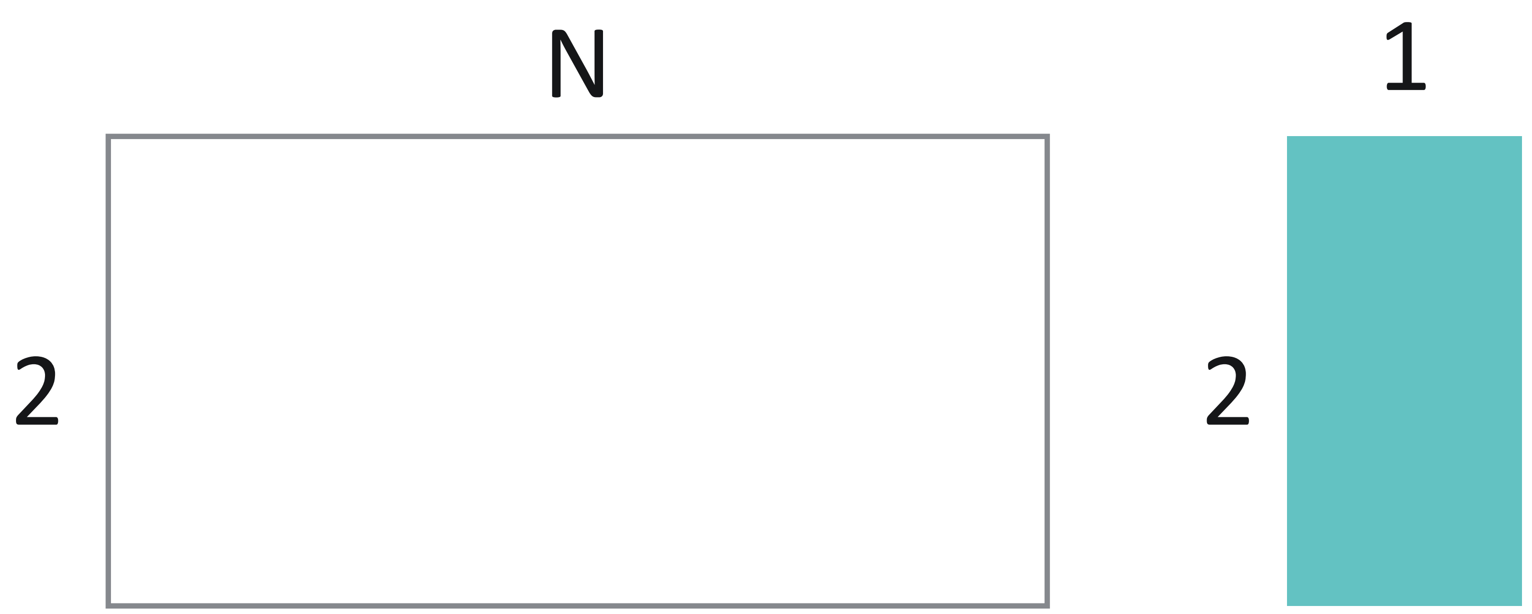
2. **부분문제로 표현하여 점화식** 구하기
   구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

3. 코드로 옮기기
   점화식을 재귀호출, 반복문 식으로 코드로 작성한다

타일채우기

문제

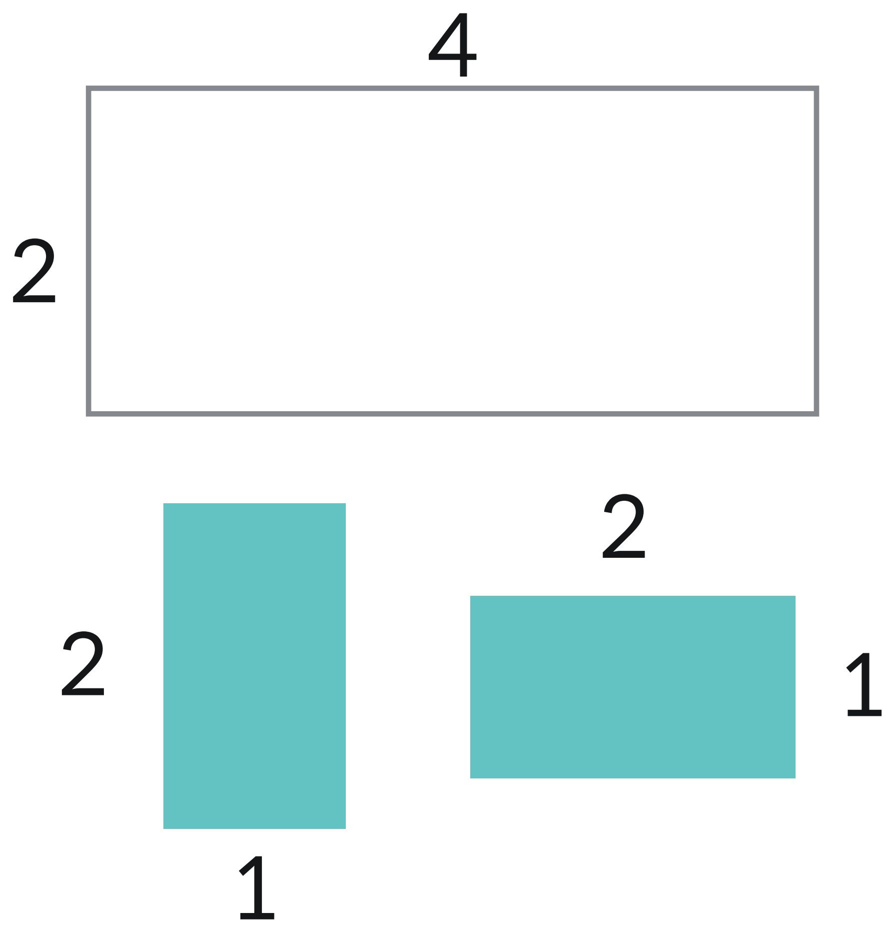
2xN짜리 액자에 2x1크기의 타일을 채워 넣으려고 합니다.
해당 액자와 타일로 만들 수 있는 **서로 다른 작품의 수는 몇 개**인가요?

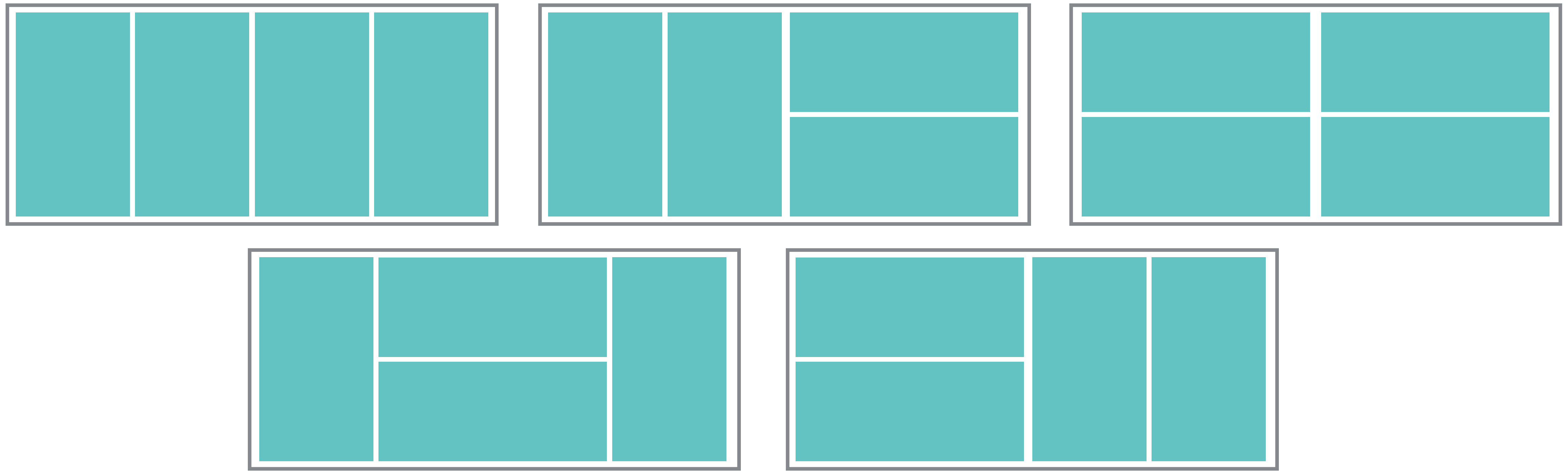


타일채우기

문제

2xN짜리 액자에 2x1크기의 타일을 채워 넣으려고 합니다.
해당 액자와 타일로 만들 수 있는 **서로 다른 타일작품의 수는 몇 개**인가요?





타일채우기

1. **구하고자 하는 값** 정의하기

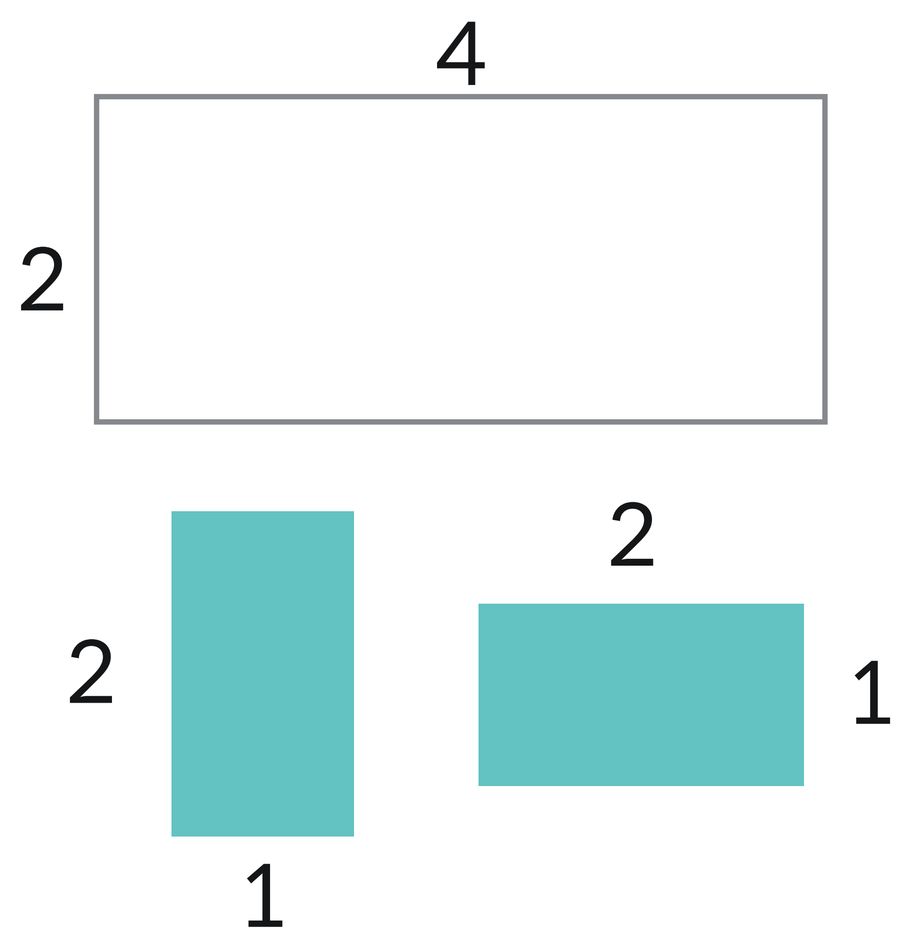
   구하고자 하는 값이 무엇인지 정의한다

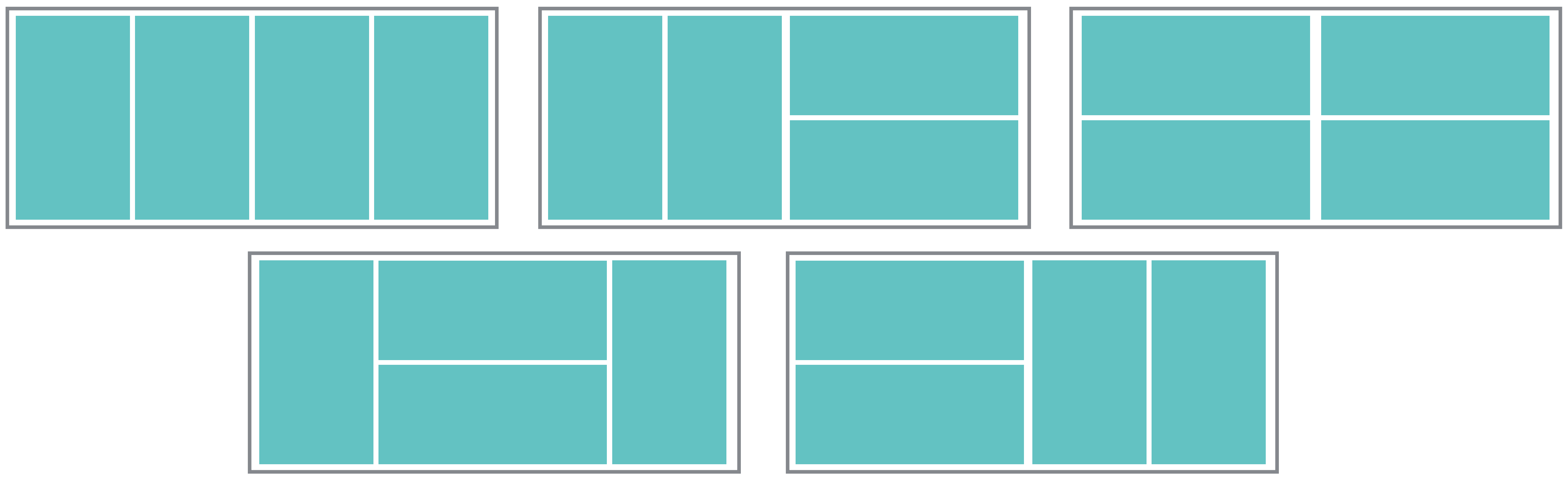
T(n) = 2 x n 크기의 액자를 채우는 경우의 수

타일채우기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

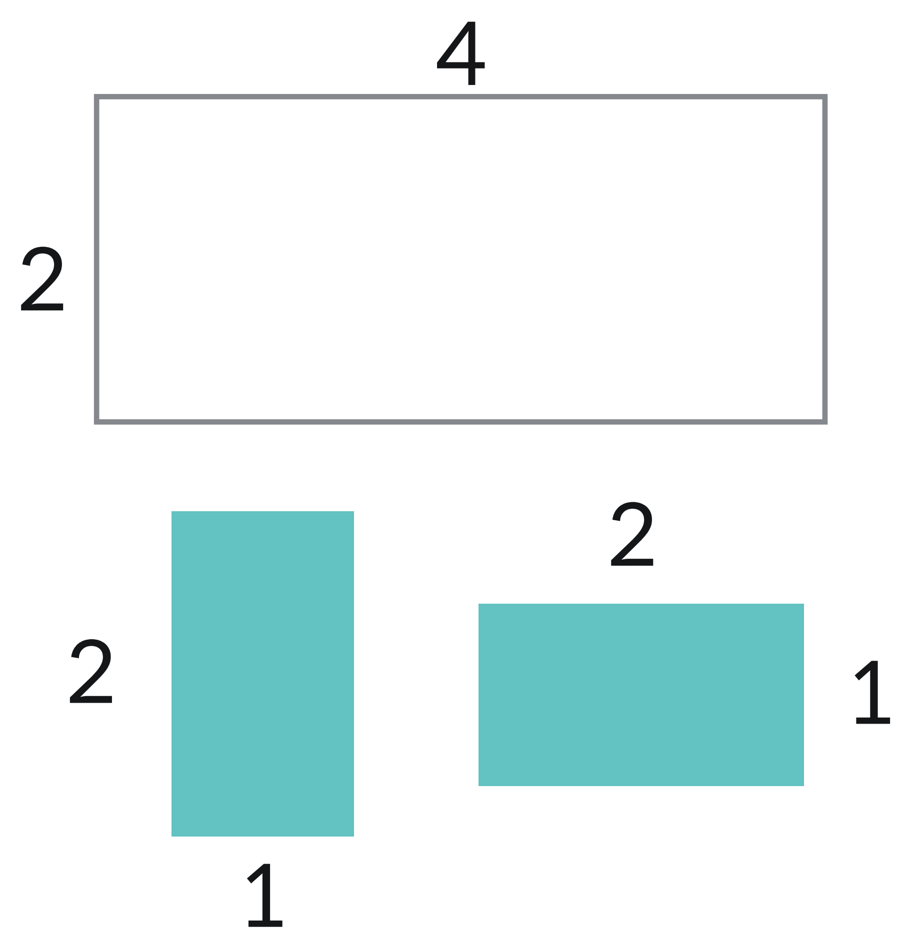


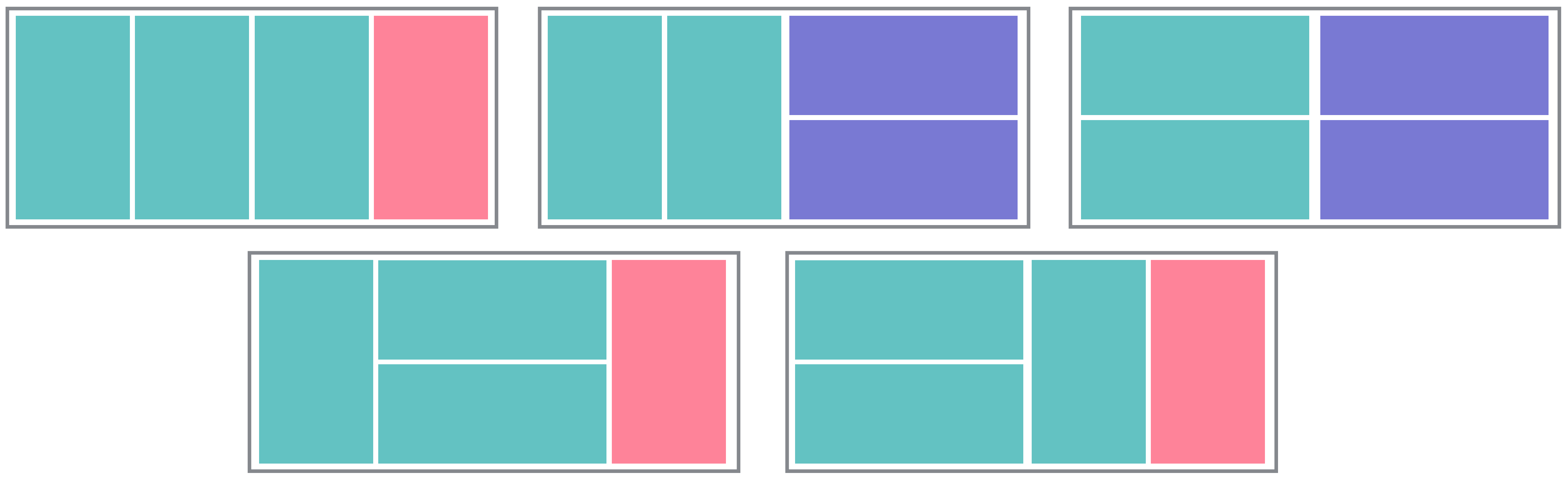


타일채우기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

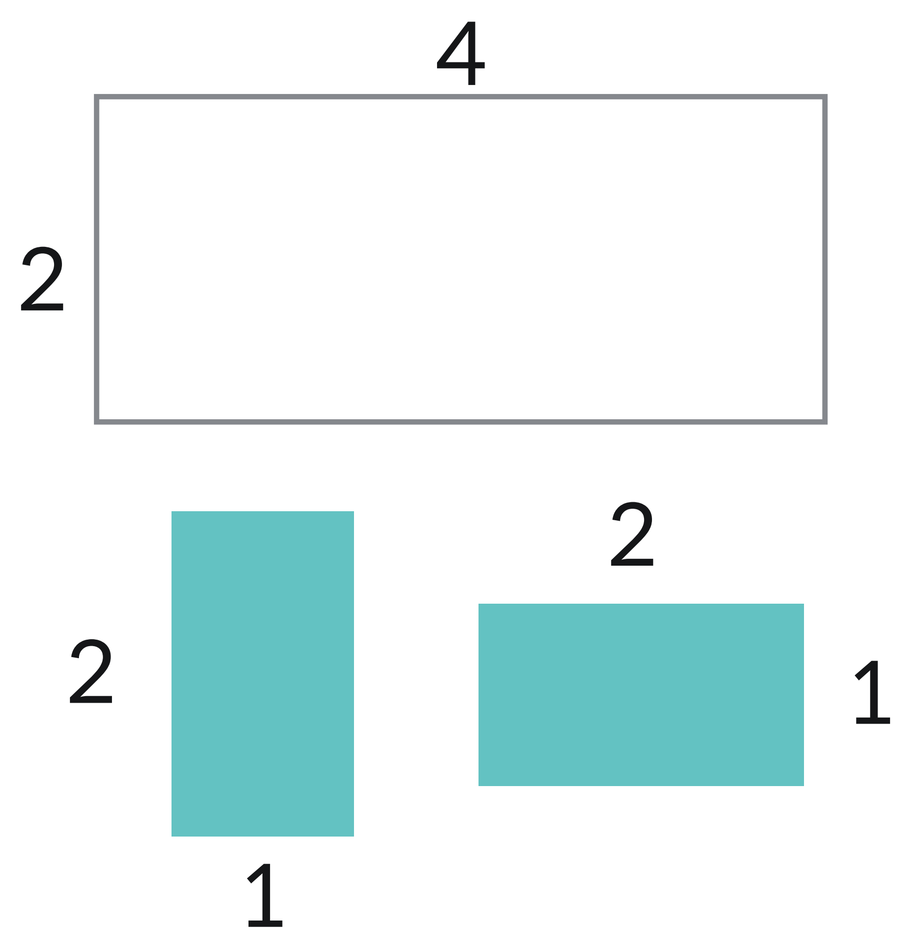


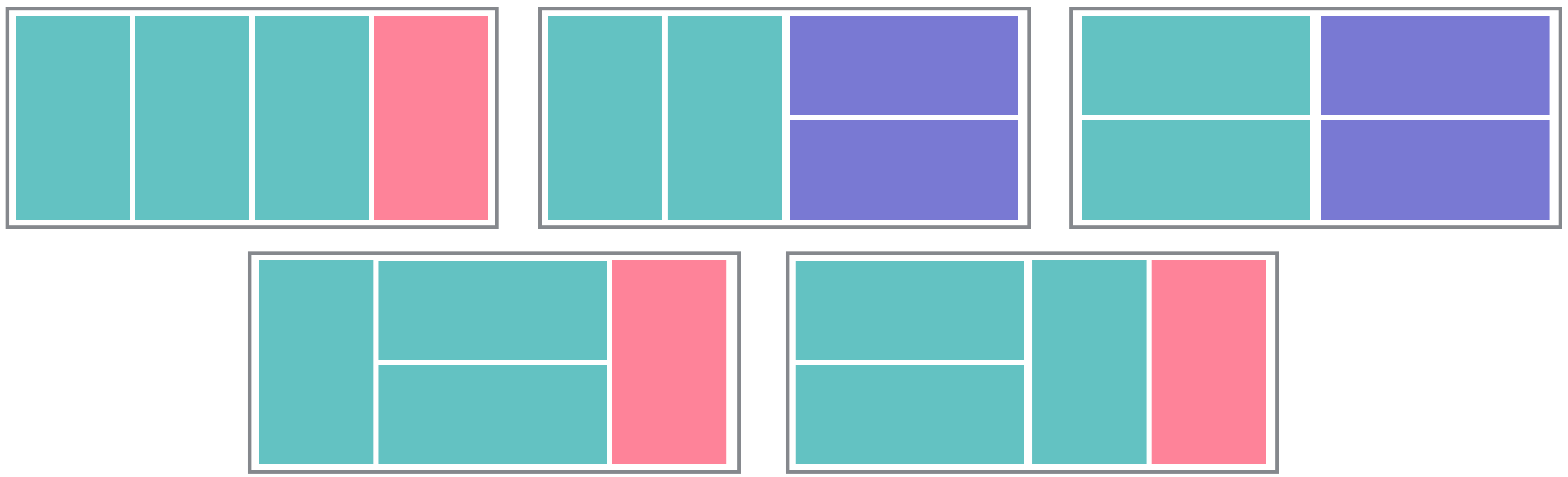


타일채우기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다





타일채우기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

1

2 x (n-1) 의 타일을 채우는 경우의 수

T(n-1)

2

2 x (n-2) 의 타일을 채우는 경우의 수

T(n-2)

타일채우기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$$T(n) = T(n-1) + T(n-2)$$

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$T(n) = T(n-1) + T(n-2)$

	0	1	2	3	4	5	6	7
T	1							

타일채우기

2. **부분문제로 표현하여 점화식** 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$T(n) = T(n-1) + T(n-2)$

	0	1	2	3	4	5	6	7
T	1	1						

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

T(n) = T(n-1) + T(n-2)

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

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$$T(n) = T(n-1) + T(n-2)$$

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

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$$T(n) = T(n-1) + T(n-2)$$

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

타일채우기

2. **부분문제로 표현하여 점화식** 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$T(n) = T(n-1) + T(n-2)$

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

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$T(n) = T(n-1) + T(n-2)$

	0	1	2	3	4	5	6	7
T	1	1	2	3	5	8	13	

타일채우기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

T(n) = 2 x n 의 액자를 채우는 경우의 수

$$T(n) = T(n-1) + T(n-2)$$

	0	1	2	3	4	5	6	7
T	1	1	2	3	5	8	13	21

타일채우기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$T(n) = T(n-1) + T(n-2)$$

```
def T(n):
    if n == 0 or n == 1: return 1

    if n in memo: return memo[n]
    else:
        memo[n] = T(n-1) + T(n-2)
    return memo[n]
```

타일채우기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$T(n) = T(n-1) + T(n-2)$$

```
T = [1, 1]
for i in range(2, n+1):
    T.append(T[i-1] + T[i-2])
```

숫자만들기

문제

1부터 M까지의 숫자를 사용하여 합이 N이 되도록 만드는 경우의 수는?
(1+2와 2+1은 서로 다른 경우입니다.)
예) 1부터 3까지 숫자를 이용해서 (M=3) 합이 5 (N=5)가 되도록 만드는 경우의 수는 모두 13가지입니다.

1 + 1 + 1 + 1+ 1	1 + 2 + 2	3 + 1 + 1
1 + 1 + 1 + 2	2 + 1 + 2	2 + 3
1 + 1 + 2 + 1	2 + 2 + 1	3 + 2
1 + 2 + 1 + 1	1 + 1 + 3	
2 + 1 + 1 + 1	1 + 3 + 1	

숫자만들기

1. **구하고자 하는 값** 정의하기

   구하고자 하는 값이 무엇인지 정의한다

sum_N(n) = 1 ~ M 의 수를 이용하여 n 를 만드는 경우의 수

숫자만들기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

1 ~ M 의 수를 이용하여 n 을 만드는 경우의 수

=

(N-1)을 만드는 경우의 수 +
(N-2)을 만드는 경우의 수 +
(N-3)을 만드는 경우의 수 +
⋯ +
(N-M)을 만드는 경우의 수

숫자만들기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

sum_N(n) = 1 ~ M 의 수를 이용하여 n을 만드는 경우의 수

sum_N(n) = sum_N(n-1) + sum_N(n-2) + ⋯ + sum_N(n-M)

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$\mathrm{sum\_N}(n) = \mathrm{sum\_N}(n-1) + \mathrm{sum\_N}(n-2) + \cdots + \mathrm{sum\_N}(n-M)$$

N=7 **M=3**

	0	1	2	3	4	5	6	7
Sum_N								

숫자만들기

2. **부분문제로 표현하여 점화식** 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

sum_N(n) = sum_N(n-1) + sum_N(n-2) + ⋯ + sum_N(n-M)

N=7 **M=3**

	0	1	2	3	4	5	6	7
Sum_N	1							

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

N=7 M=3

	0	1	2	3	4	5	6	7
Sum_N	1	1						

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

N=7 **M=3**

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

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

sum_N(n) = sum_N(n-1) + sum_N(n-2) + ⋯ + sum_N(n-M)

N=7 **M=3**

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

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

sum_N(n) = sum_N(n-1) + sum_N(n-2) + ··· + sum_N(n-M)

N=7 **M=3**

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

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

N=7 **M=3**

	0	1	2	3	4	5	6	7
Sum_N	1	1	2	4	7	13		

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

N=7 **M=3**

	0	1	2	3	4	5	6	7
Sum_N	1	1	2	4	7	13	24	

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

sum_N(n) = sum_N(n-1) + sum_N(n-2) + ⋯ + sum_N(n-M)

N=7 M=3

	0	1	2	3	4	5	6	7
Sum_N	1	1	2	4	7	13	24	44

숫자만들기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

```
def sum_N(n):
    if n == 0: return 1
    if n in memo:
        return memo[n]
```

```
else:
    sum_temp = 0
    for i in range(1, M+1)
            sum_temp += sum_N(n-i)
        memo[n] = sum_temp
    return memo[n]
```

숫자만들기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

```
sum_N= [1]
for n in range(1, N+1):
    sum_temp = 0
    for i in range(1, M+1)
        sum_temp += sum_N[n-i]
    sum_N.append(sum_temp)
```

숫자만들기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$sum\_N(n) = sum\_N(n-1) + sum\_N(n-2) + \cdots + sum\_N(n-M)$$

```
sum_N= [1]
for n in range(1, N+1):
    sum_temp = 0
    for i in range(1, min(n,M)+1):
        sum_temp += sum_N[n-i]
    sum_N.append(sum_temp)
```

짜장, 짬뽕, 볶음밥!

문제

상훈이는 **점심엔 늘 짜장, 짬뽕, 볶음밥 셋 중 하나**를 먹는다. 하지만 거기엔 규칙이 하나 있는데, **전날 먹은 음식은 먹지 않는다**는 것이다. 예를 들어 어제 짜장면을 먹었으면, 오늘은 짬뽕이나 볶음밥 중 하나를 먹어야 하는 것이다.

짜장, 짬뽕, 볶음밥 **선호도는 그날그날 다른데**, 매일 선호도가 주어질 때, 적절히 날마다 음식을 결정하여 **총선호도의 최대값**을 구하시오.

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

총 선호도 116

숫자만들기

1. **구하고자 하는 값** 정의하기

   구하고자 하는 값이 무엇인지 정의한다

Delicious(i) = i번째 날까지 구한 최대 선호도 합

숫자만들기

1. **구하고자 하는 값** 정의하기

구하고자 하는 값이 무엇인지 정의한다

짜장: 0, 짬뽕: 1, 볶음밥: 2

Delicious(i, j) = i번째 날에 j음식을 먹었을 때 지금까지의 최대 선호도합

숫자만들기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

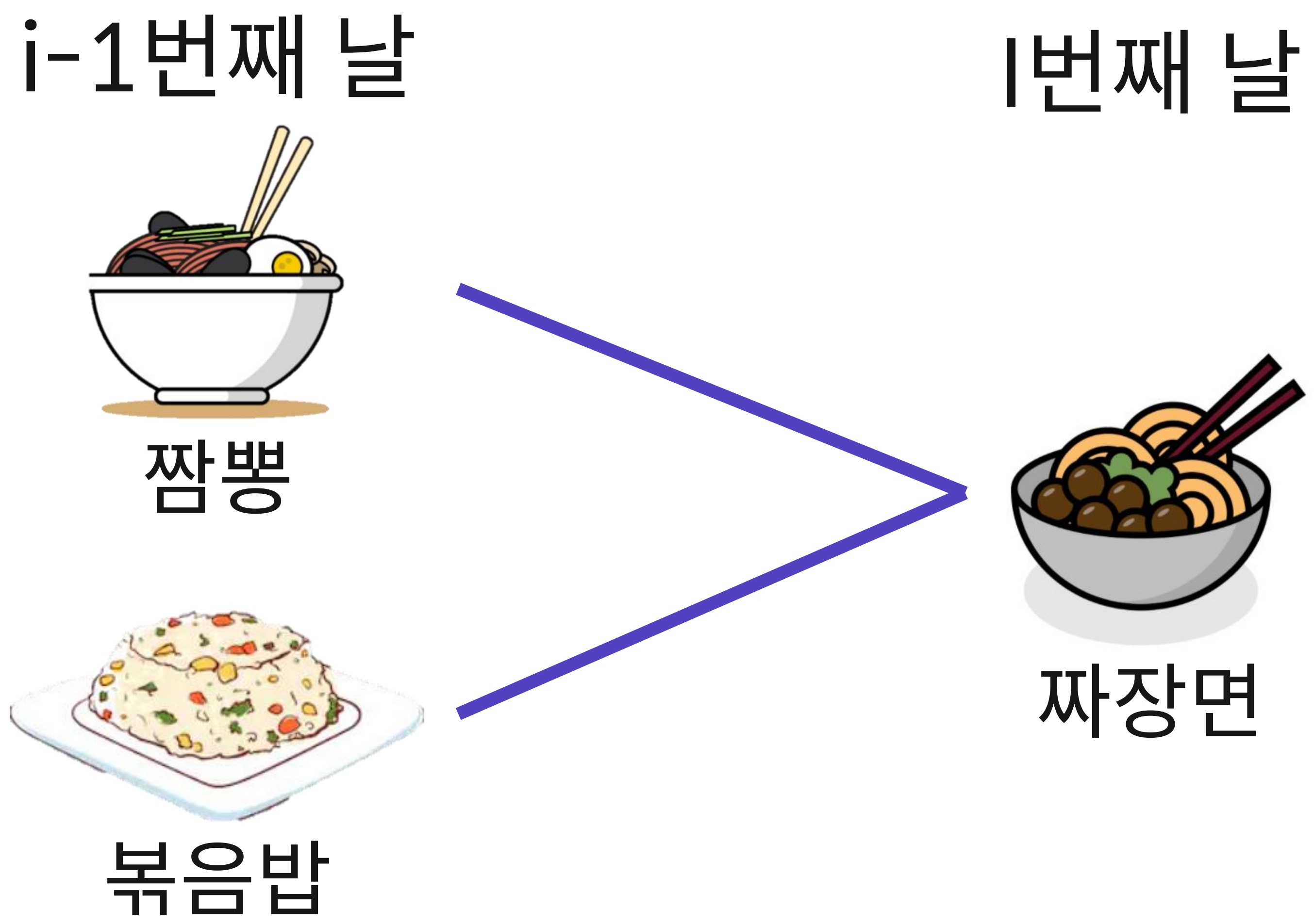
1번째 날

짜장면

숫자만들기

2. 부분문제로 표현하여 점화식 구하기

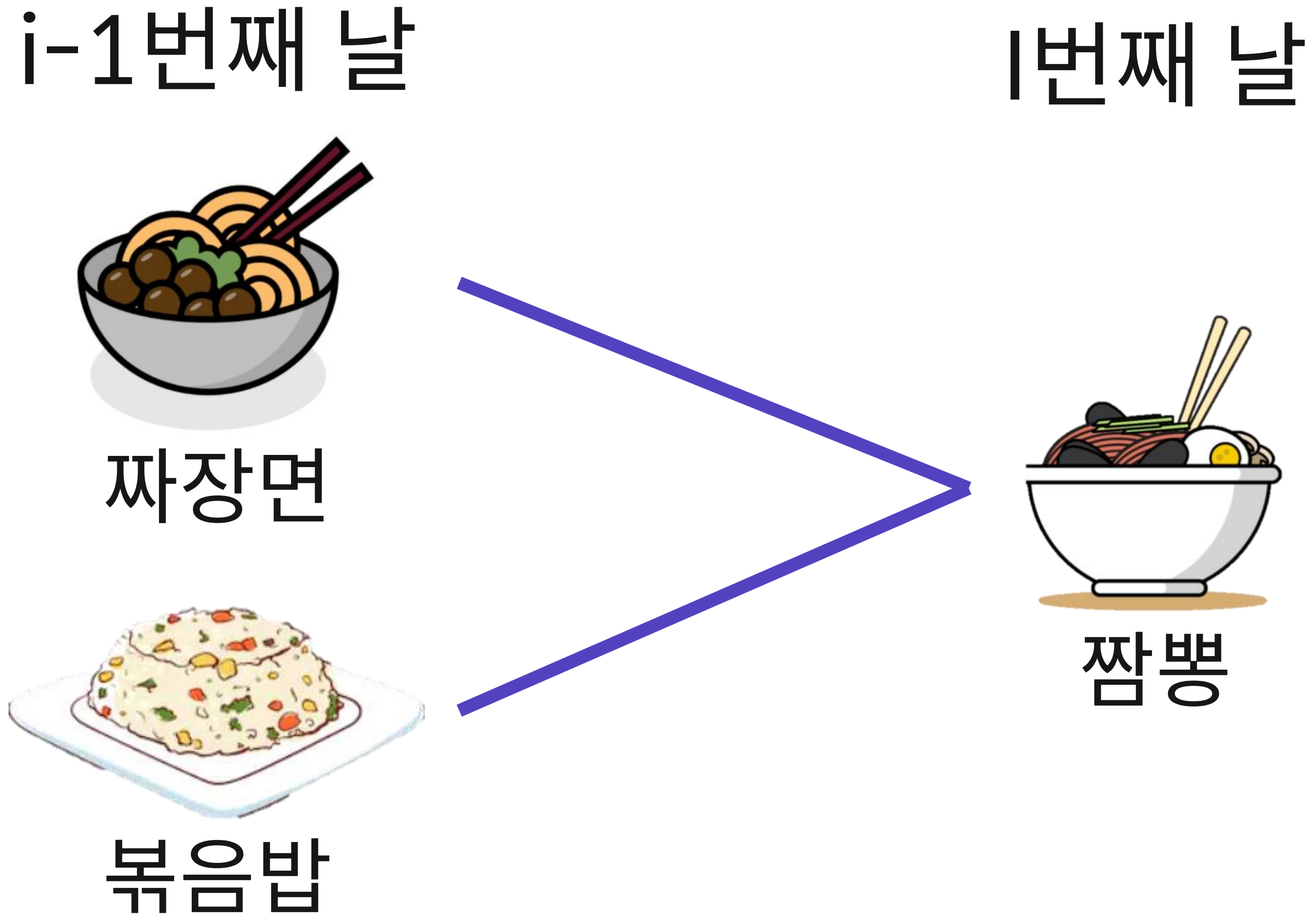
구하고자 하는 값을 부분문제로 구성된 식으로 표현한다



숫자만들기

2. 부분문제로 표현하여 점화식 구하기

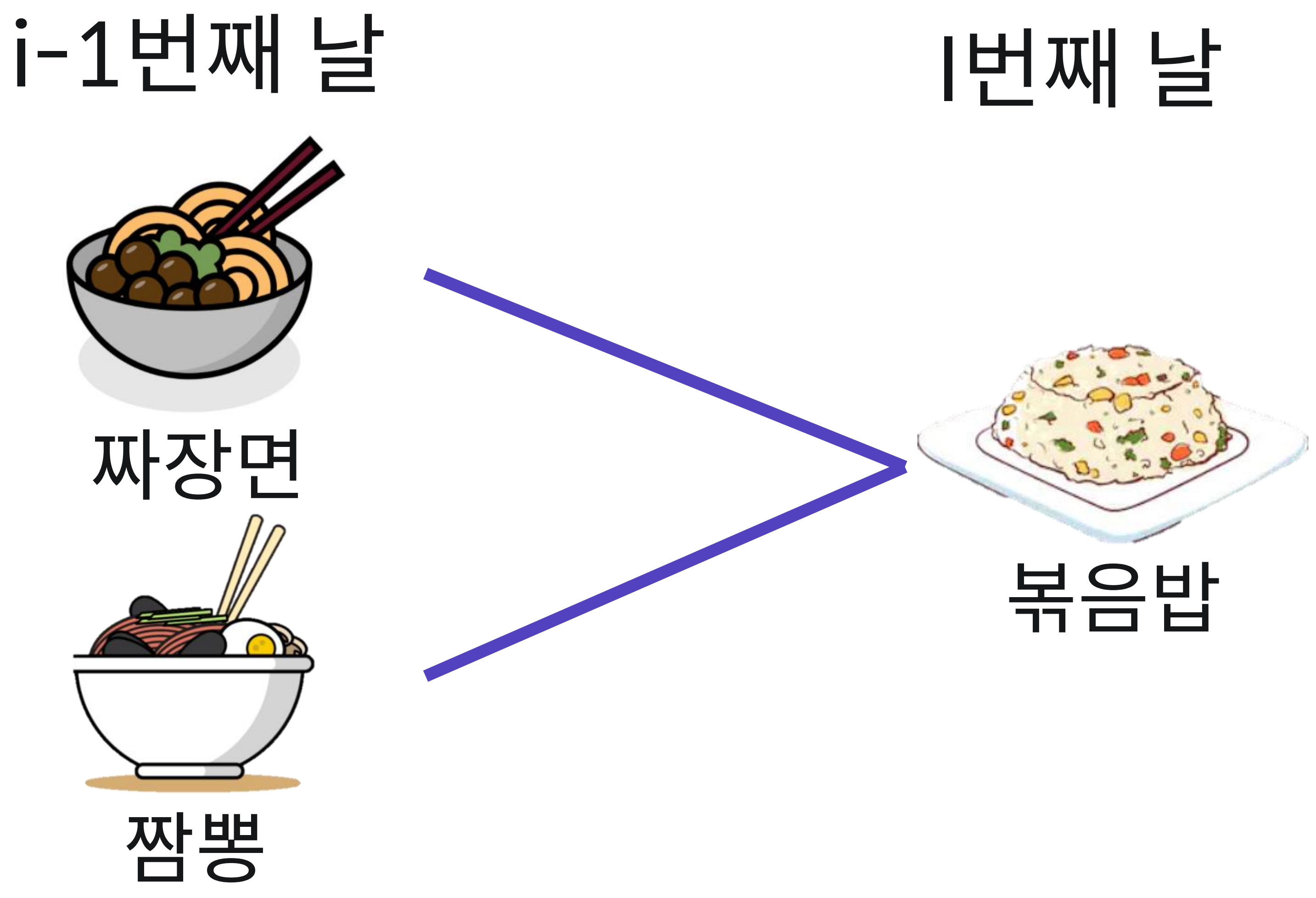
구하고자 하는 값을 부분문제로 구성된 식으로 표현한다



숫자만들기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다



숫자만들기

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$\text{Delicious}(i, j) = \max(\text{Delicious}(i-1, k)) + \text{preference}(i, j)\ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일			
2일			
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27		
2일			
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j)\ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	
2일			
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \; \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일			
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = max(Delicious(i-1, k)) + preference(i, j) \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53		
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	37
3일			

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	37
3일	78		

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$\text{Delicious}(i, j) = \max(\text{Delicious}(i-1, k)) + \text{preference}(i, j)\ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	37
3일	78	75	

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j) \ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	37
3일	78	75	116

숫자만들기

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j)\ \{k \neq j\}$$

preference

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	18	36	10
3일	7	22	45

Delicious

구분	짜장	짬뽕	볶음밥
1일	27	8	35
2일	53	71	37
3일	78	75	116

숫자만들기

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

$$Delicious(i, j) = \max(Delicious(i-1, k)) + preference(i, j)\ \{k \neq j\}$$

```
for i in range(1, days+1):
    for j in range(3):
        for k in range(3):
            if j == k: continue
            Delicious[i][j] = max(Delicious[i][j], Delicious[i-1][k])
        Delicious[i][j] += preference[i][j]
```

연속 부분 최대합

문제

주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.

1	2	-4	5	3	-2	9	10

연속 부분 최대합 - 완전탐색법

문제

주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.

연속 부분 최대합 - 완전탐색법

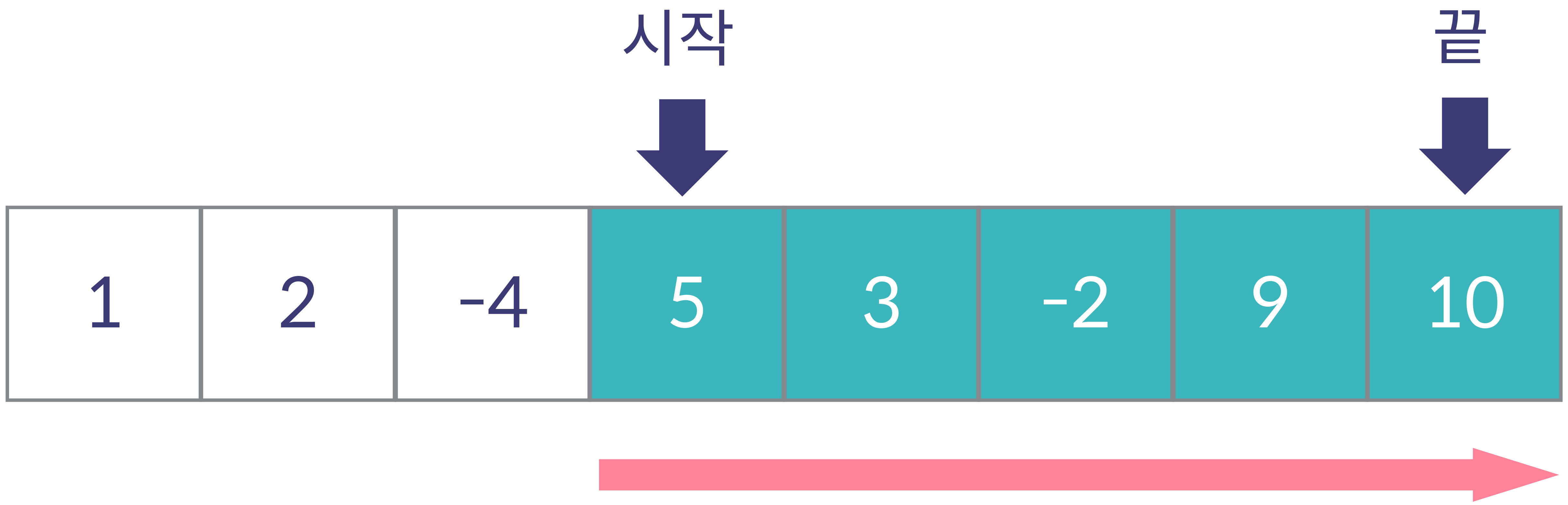
문제

주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.

연속 부분 최대합 - 완전탐색법

문제

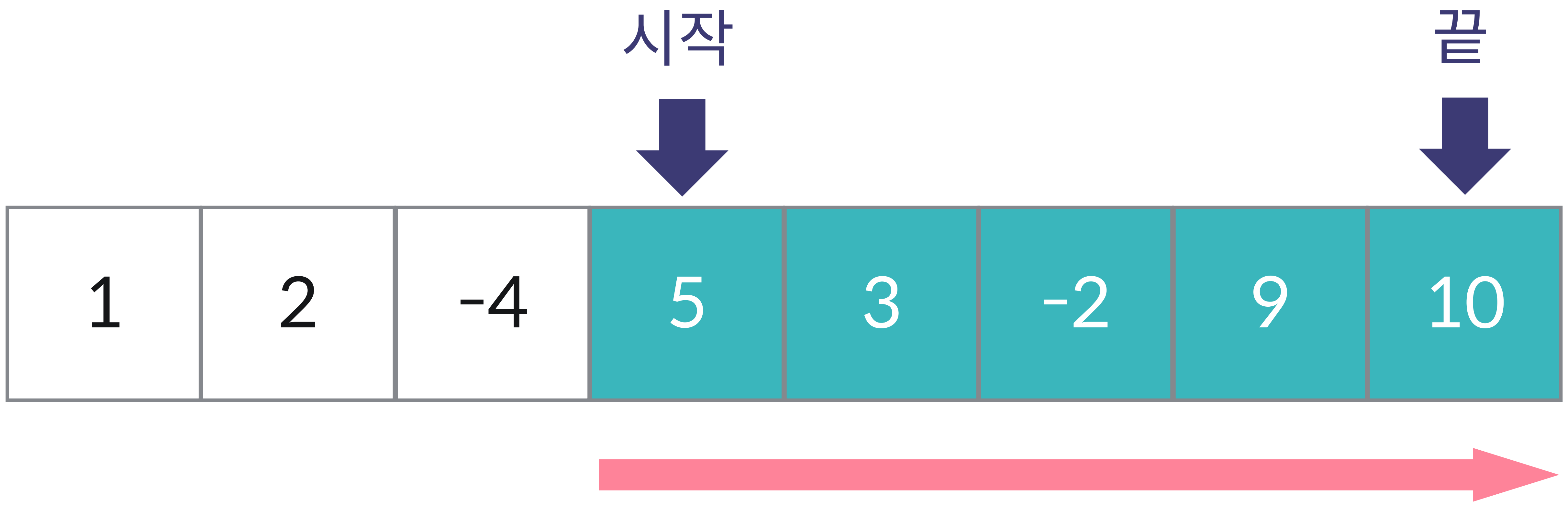
주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.



연속 부분 최대합 - 완전탐색법

문제

주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.



$o(n^3)$

연속 부분 최대합 – 분할정복법

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

1. 자른 부분의 **왼쪽**에 최대부분합이 존재하는 경우
2. 자른 부분의 **오른쪽**에 최대부분합이 존재하는 경우
3. **자른 부분을 걸쳐서** 최대부분합이 존재하는 경우

연속 부분 최대합 – 분할정복법

문제

주어진 수열에서 **연속하는 부분**을 선택하여, **그 최대합**을 구하세요.

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

연속 부분 최대합 – 분할정복법

1 자른 부분의 **왼쪽**에 최대부분합이 존재하는 경우

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

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

연속 부분 최대합 – 분할정복법

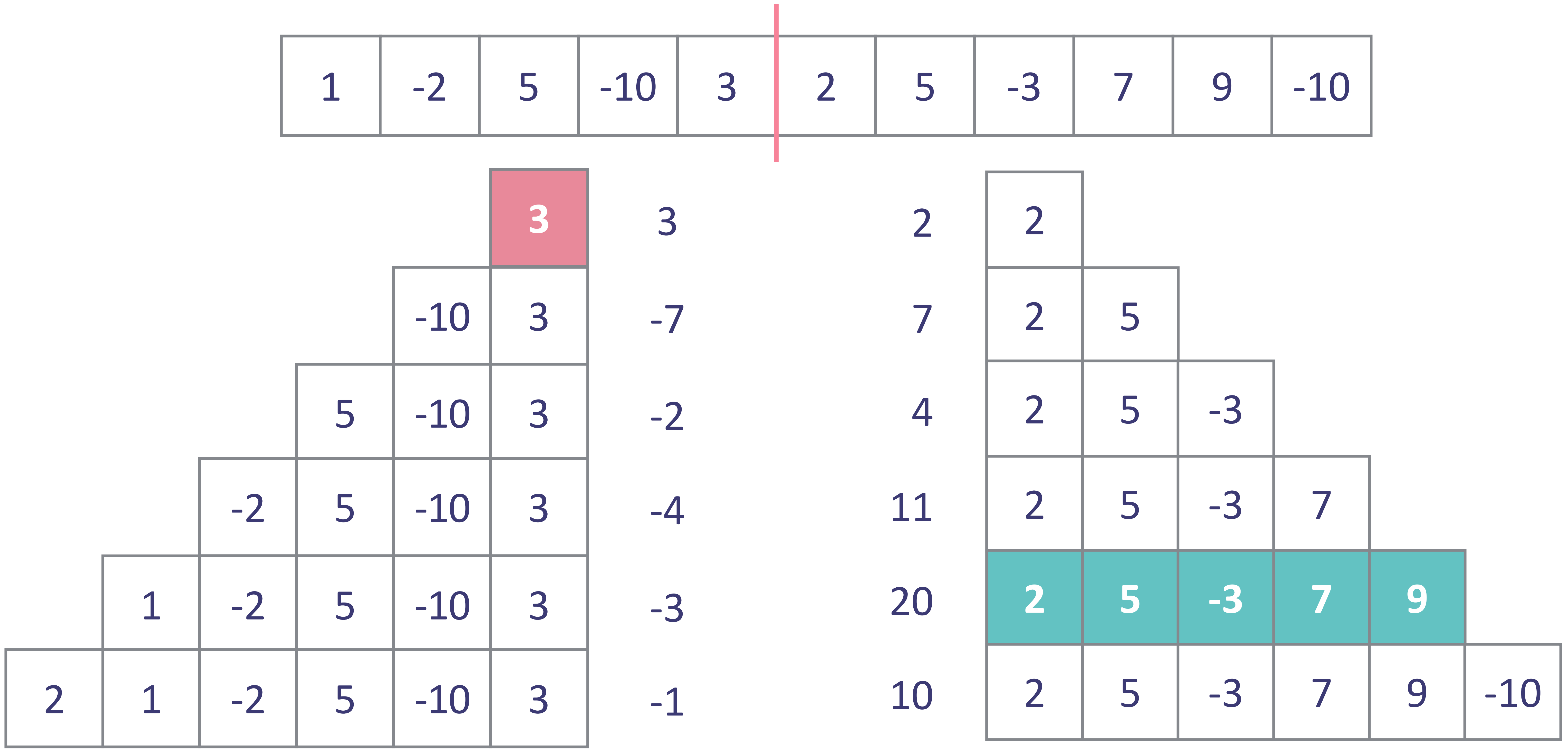
2 자른 부분의 오른쪽에 최대부분합이 존재하는 경우

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

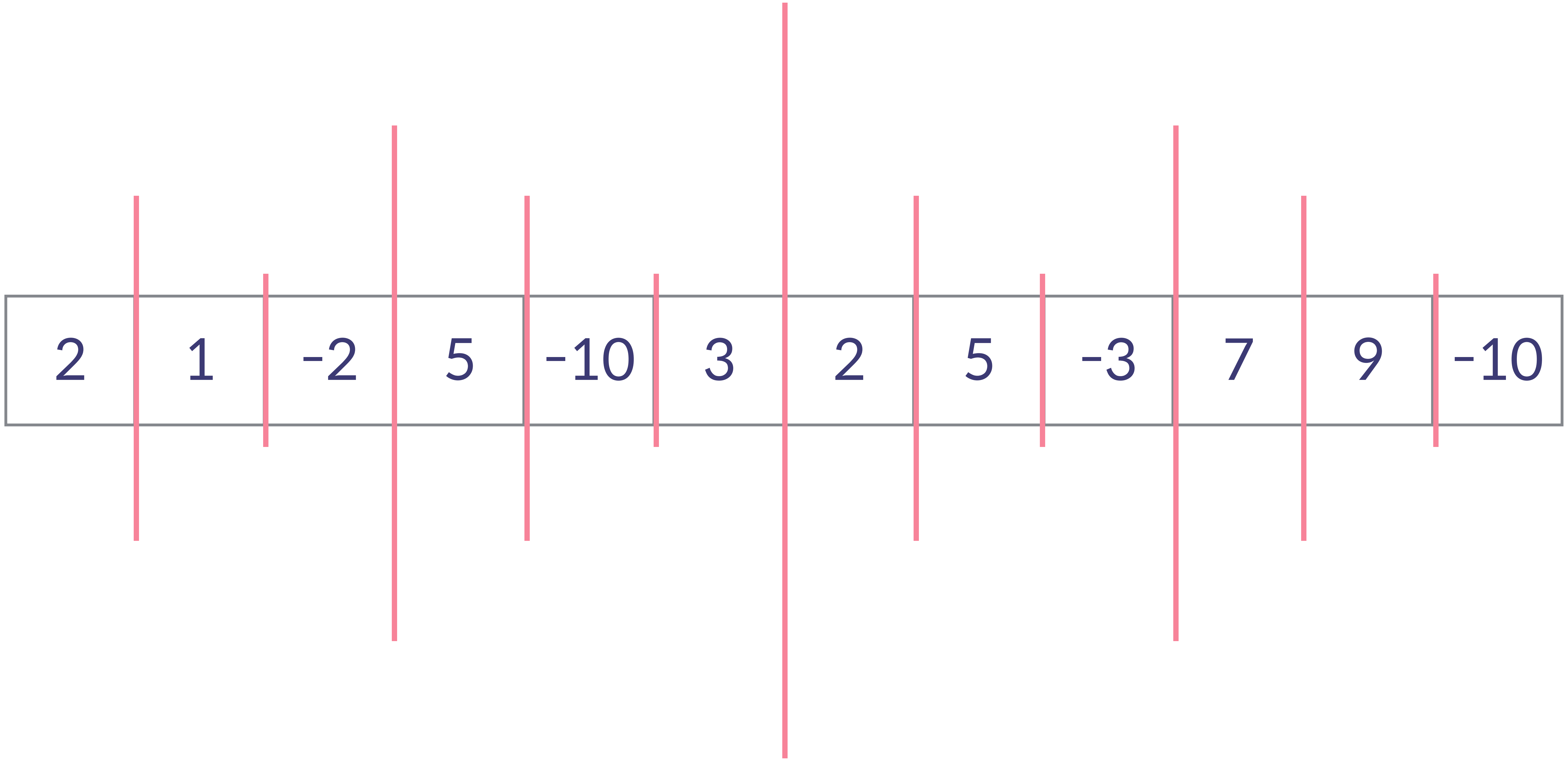
2	2					
7	2	5				
4	2	5	-3			
11	2	5	-3	7		
20	2	5	-3	7	9	
10	2	5	-3	7	9	-10

연속 부분 최대합 – 분할정복법

3 자른 부분을 걸쳐서 최대부분합이 존재하는 경우



연속 부분 최대합 – 분할정복법



절반씩 잘라서 확인하기 때문에 분할하는데 $O(\log(n))$ 이 걸립니다

연속 부분 최대합 – 분할정복법

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

한번 잘랐을 때 양쪽방향으로 원소를 하나씩 돌기 때문에 $O(n)$ 이 걸립니다.

이러한 과정을 총 $\log n$번 반복하기 때문에
시간복잡도는 $O(n \log n)$입니다.

연속 부분 최대합 – 동적계획법

1. **구하고자 하는 값** 정의하기

   구하고자 하는 값이 무엇인지 정의한다

maxSum(n) = n번째 숫자를 가장 오른쪽으로 하는 연속 부분 최대합

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

1. 직전 원소를 마지막으로 하는 최대연속 합에 현재 n번째 값을 포함하기
2. n번째부터 새롭게 연속합을 시작하기

연속 부분 최대합 – 동적계획법

2. **부분문제로 표현하여 점화식** 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$\text{maxSum}(n) = \max(\underline{\text{maxSum}(n-1)}, 0) + \underline{\text{arr}(n)}$$

1 직전 원소를 마지막으로 하는 최대 연속합에
현재 n번째 값을 포함하기

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$\text{maxSum}(n) = \max(\text{maxSum}(n-1), \underline{0}) + \underline{\text{arr}(n)}$$

2 n번째부터 새롭게 연속합을 시작하기

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$maxSum(n) = max(maxSum(n-1), 0) + arr(n)$$

	0	1	2	3	4	5	6	7
arr	1	2	-4	5	3	-2	9	10
T	1							

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$maxSum(n) = max(maxSum(n-1), 0) + arr(n)$$

	0	1	2	3	4	5	6	7
arr	1	2	-4	5	3	-2	9	10
T	1	3						

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

maxSum(n) = max(maxSum(n-1), 0) + arr(n)

arr	1	2	-4	5	3	-2	9	10
	0	1	2	3	4	5	6	7
T	1	3	-1					

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$maxSum(n) = max(maxSum(n-1), 0) + arr(n)$$

	0	1	2	3	4	5	6	7
arr	1	2	-4	5	3	-2	9	10
T	1	3	-1	5				

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$maxSum(n) = max(maxSum(n-1), 0) + arr(n)$$

arr	1	2	-4	5	3	-2	9	10
	0	1	2	3	4	5	6	7
T	1	3	-1	5	8			

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

maxSum(n) = max(maxSum(n-1), 0) + arr(n)

arr	1	2	-4	5	3	-2	9	10
	0	1	2	3	4	5	6	7
T	1	3	-1	5	8	6		

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

maxSum(n) = max(maxSum(n-1), 0) + arr(n)

	0	1	2	3	4	5	6	7
arr	1	2	-4	5	3	-2	9	10
T	1	3	-1	5	8	6	15	

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

$$maxSum(n) = max(maxSum(n-1), 0) + arr(n)$$

	0	1	2	3	4	5	6	7
arr	1	2	-4	5	3	-2	9	10
T	1	3	-1	5	8	6	15	25

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

arr	1	2	-4	5	3	-2	9	10

	0	1	2	3	4	5	6	7
T	1	3	-1	5	8	6	15	25

1~N번째를 마지막으로 최대 연속 부분합을 구성하는 것 중 **최대**를 찾는다

연속 부분 최대합 – 동적계획법

2. 부분문제로 표현하여 점화식 구하기 - 동작확인

구하고자 하는 값을 부분문제로 구성된 식으로 표현한다

arr								
	1	2	-4	5	3	-2	9	10

	0	1	2	3	4	5	6	7
T	1	3	-1	5	8	6	15	25

1~N번째를 마지막으로 최대 연속 부분합을 구성하는 것 중 **최대**를 찾는다

연속 부분 최대합 – 동적계획법

3. 코드로 옮기기

점화식을 재귀호출, 반복문 식으로 코드로 작성한다

maxSum(n) = max(maxSum(n-1), 0) + arr(n)

```
maxSum = []
final_max = -float('inf')
maxSum.append(max(0,arr[0]))
for n in range(1, N):
    maxSum.append(max(maxSum[n-1], 0) + arr[n])
    if final_max < maxSum[n]:
        final_max = maxSum[n]
```

07 정리

동적계획법은…

- 복잡한 문제를 작은 하위문제로 나누어 푸는 방식입니다. 중복되는 하위문제를 한번만 풂으로써 시간복잡도면에서 이익이 있습니다.

- 동적계획법으로 풀 수 있는 문제는
중복되는 부분문제로 이루어져있고 **최적 부분 구조**를 만족합니다.

- 동적계획법을 구현하기 위해서
Top-down (재귀식 방법)과 Buttom-up (반복문식 방법)을 활용할 수 있습니다.

Credit

/* elice */

코스 매니저
장석준

콘텐츠 제작자
임도연

강사
임도연

디자인
강혜정

Contact

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io