재귀(순환)

재귀 알고리즘

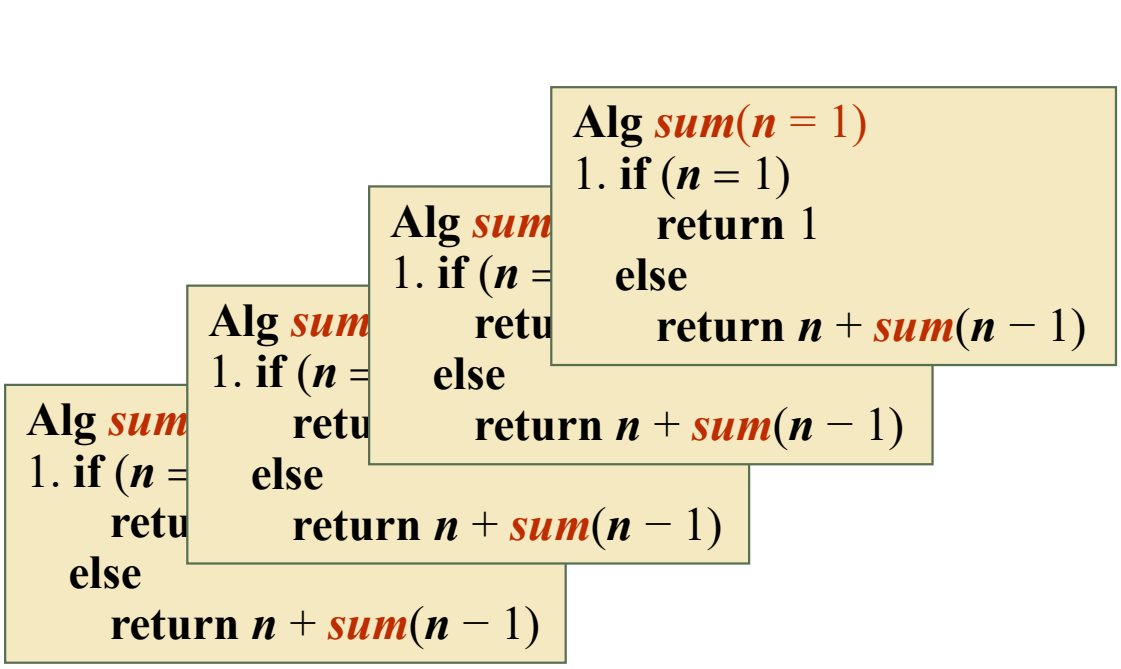
- 알고리즘 자신을 사용하여 정의된 알고리즘을 **재귀적**(recursive)이라고 말한다
  - **비재귀적**(nonrecursive) 또는 **반복적**(iterative) 알고리즘과 대조

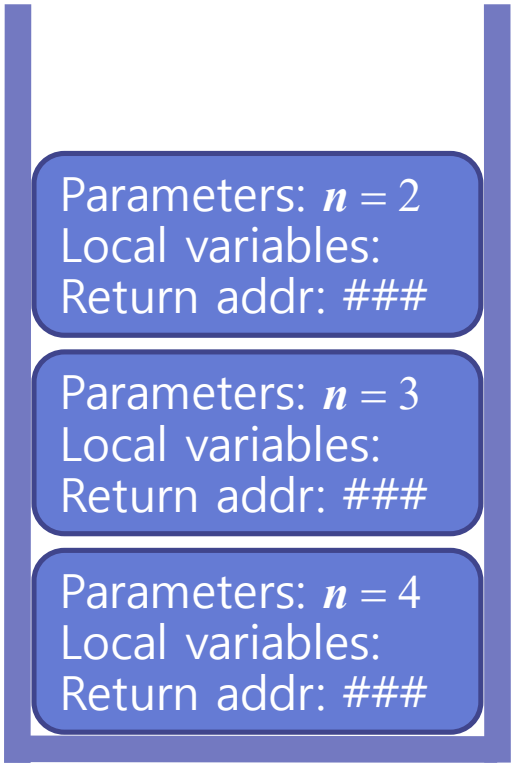
- 재귀의 요소
  - **재귀 케이스**(recursion): 차후의 재귀호출은 **작아진** 부문제들(subproblems)을 대상으로 이루어진다
  - **베이스 케이스**(base case): 부문제들이 충분히 작아지면, 알고리즘은 재귀를 사용하지 않고 이들을 직접 해결한다

```
Alg sum(n)
1. if (n = 1)                    {base case}
       return 1
   else                          {recursion}
       return n + sum(n − 1)
```

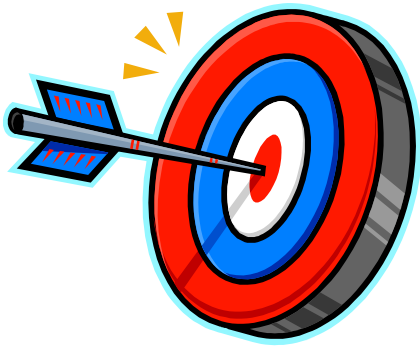
작동 원리

- 보류된 재귀호출(즉, 시작했지만 완료하지 않고 대기중인 호출들)을 위한 변수들에 관련된 저장/복구는 컴퓨터에 의해 자동적으로 수행





기본 규칙



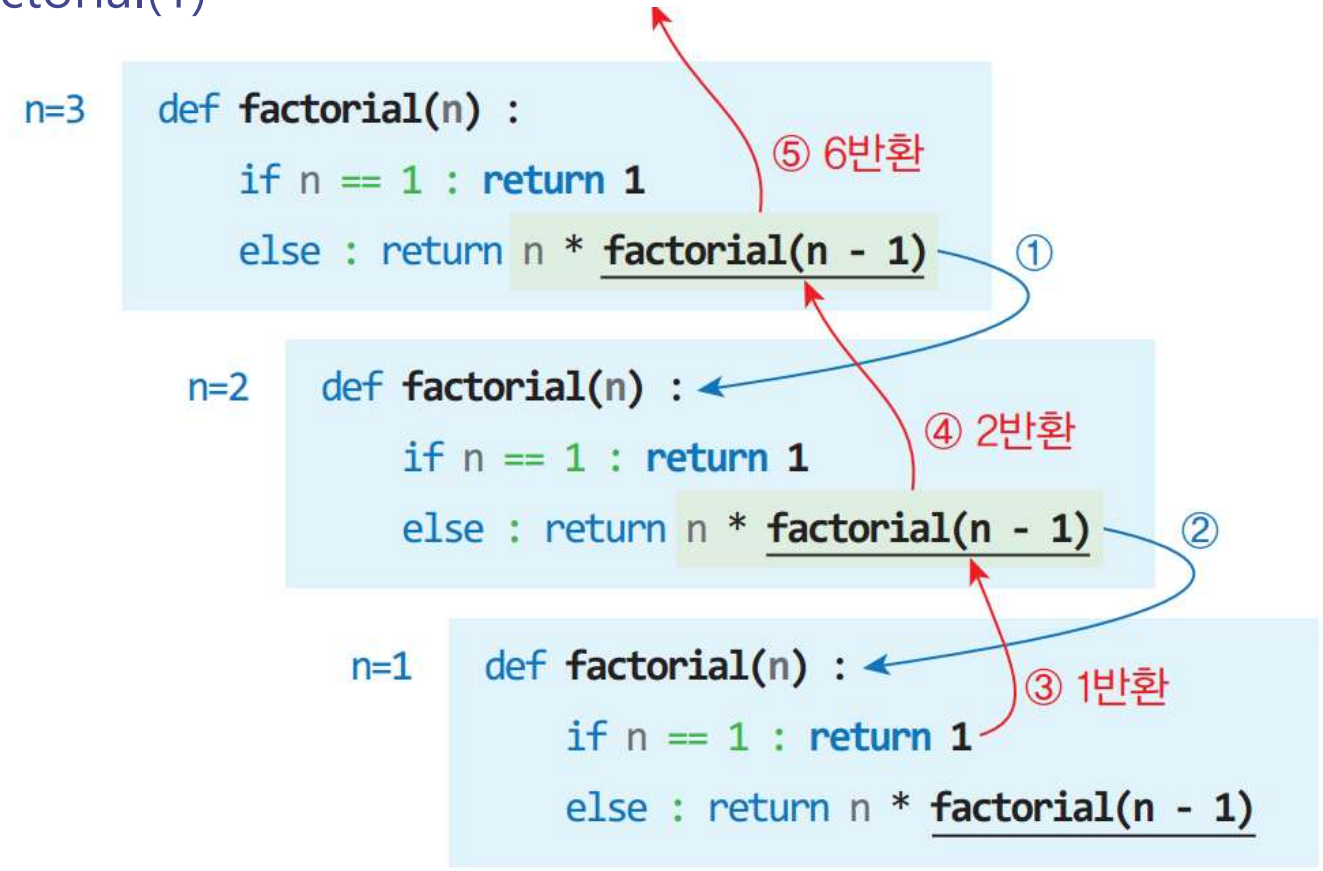
- **베이스 케이스**
  - 베이스 케이스를 항상 가져야 하며, 이 부분은 재귀 없이 해결 가능
- **진행 방향**
  - 재귀적으로 해결되어야 할 경우, 재귀호출은 항상 베이스 케이스를 향하는 방향으로 진행
- **정상작동 가정**
  - 모든 재귀호출이 제대로 작동한다고 가정!
- **적절한 사용**
  - 꼭 필요할 때만 사용 – 저장/복구 때문에 성능 저하

팩토리얼 구하기

$$n! = \begin{cases} 1 & n=1 \\ n*(n-1)! & n>1 \end{cases}$$

- 순환적인 함수 호출 순서

factorial(3) = 3 * factorial(2)
= 3 * 2 * factorial(1)
= 3 * 2 * 1
= 3 * 2
= 6



팩토리얼: 순환과 반복

- n의 팩토리얼 구하기

순환 구조		반복 구조
$n!=n*(n-1)!$	↔	$n!=n*(n-1)*(n-2)*\cdots*1$

- 순환(recursion): $O(n)$
  - 순환적인 문제에서는 자연스러운 방법
  - 함수 호출의 오버헤드
- 반복(iteration): $O(n)$
  - for나 while문 이용. 수행속도가 빠름.
  - 순환적인 문제에서는 프로그램 작성이 어려울 수도 있음.

- 대부분의 순환은 반복으로 바꾸어 작성할 수 있음

순환이 더 빠른 예: 거듭제곱 계산

- 방법 1: 반복 구조

```
def power_iter(x, n) :                  # 반복으로 xn을 구하는 함수
    result = 1.0
    for i in range(n):                  # 루브: n번 반복
        result = result * x
    return result
```

  - 내부 반복문 : $O(n)$

순환적인 거듭제곱 함수

- 방법 2: 순환 구조

```
power(x, n)

if n = 0
    then return 1;
else if n이 짝수
    then return power(x2, n/2);
else if n이 홀수
    then return x*power(x2, (n-1)/2);
```

```
def power(x, n) :
    if n == 0 : return 1
    elif (n % 2) == 0 :                    # n이 짝수
        return power(x*x, n//2)            # 정수의 나눗셈
    else :
        return x * power(x*x, (n-1)//2)    # n이 홀수
```

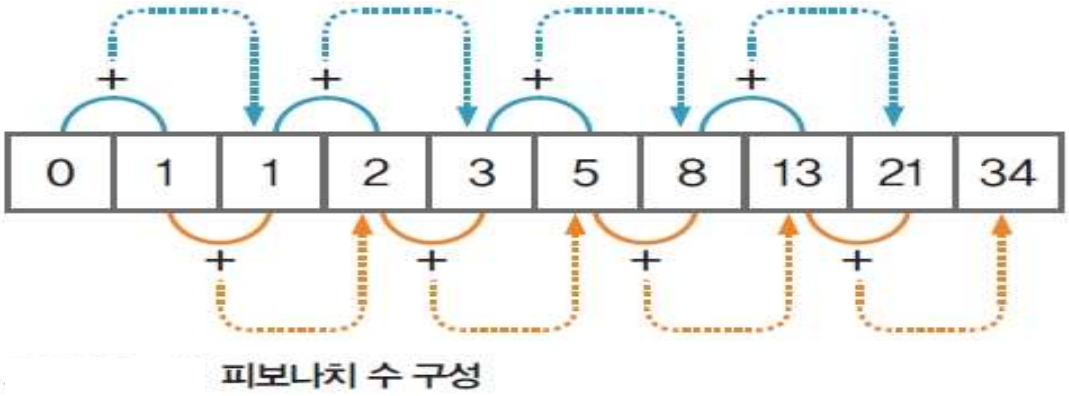
복잡도 분석

- 시간 복잡도
  - 순환적인 함수: $O(log_2 n)$
  - 반복적인 함수: $O(n)$

순환이 느린 예: 피보나치 수열

- 순환 호출을 사용하면 비효율적인 예
- 피보나치 수열: 0,1,1,2,3,5,8,13,21,...

$$fib(n)\begin{cases} 0 & n=0 \\ 1 & n=1 \\ fib(n-2)+fib(n-1) & otherwise \end{cases}$$



피보나치 수 구성

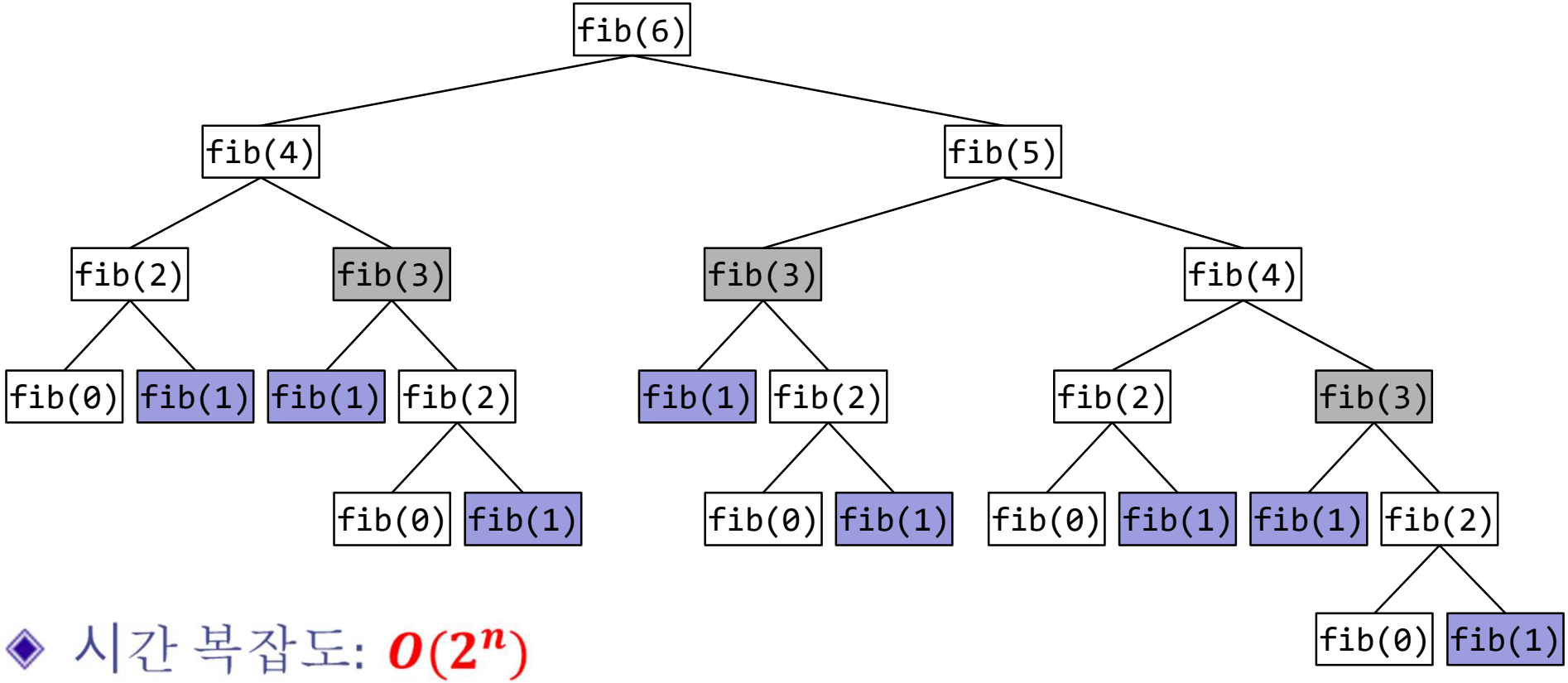
순환이 느린 예: 피보나치 수열

- 순환적인 구현

```
def fib(n) :                                    # 순환으로 구현한 피보나치 수열
    if n == 0 : return 0                        # 종료조건
    elif n == 1 : return 1                      # 종료조건
    else :
        return fib(n - 1) + fib(n - 2)          # 순환호출
```

순환적인 피보나치의 비효율성

- 같은 항이 중복해서 계산됨!
  - n이 커지면 더욱 심각



- 시간 복잡도: $O(2^n)$

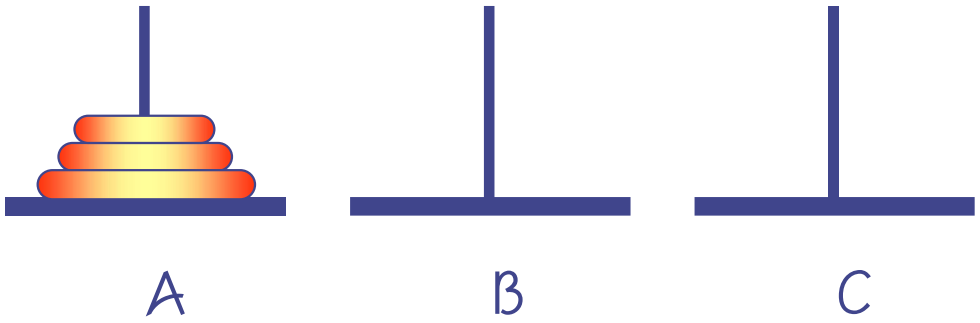
반복적인 피보나치 수열

```
def fib_iter(n) :                    # 반복으로 구현한 피보나치 수열
    if (n < 2): return n

    last = 0
    current = 1
    for i in range(2, n+1) :         # 반복 루프
        tmp = current
        current += last
        last = tmp
    return current
```

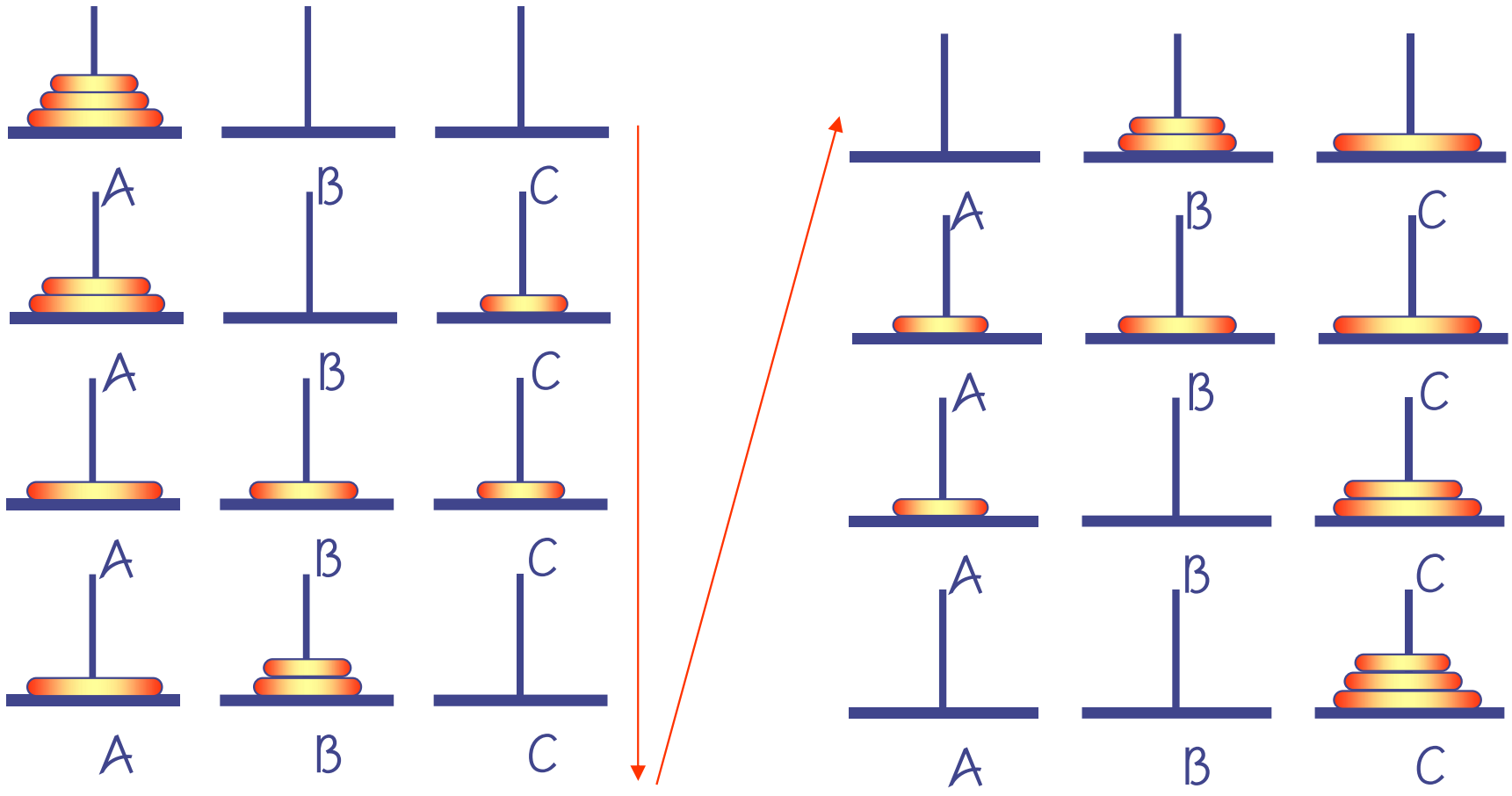
- 시간 복잡도: $O(n)$

하노이 탑 문제

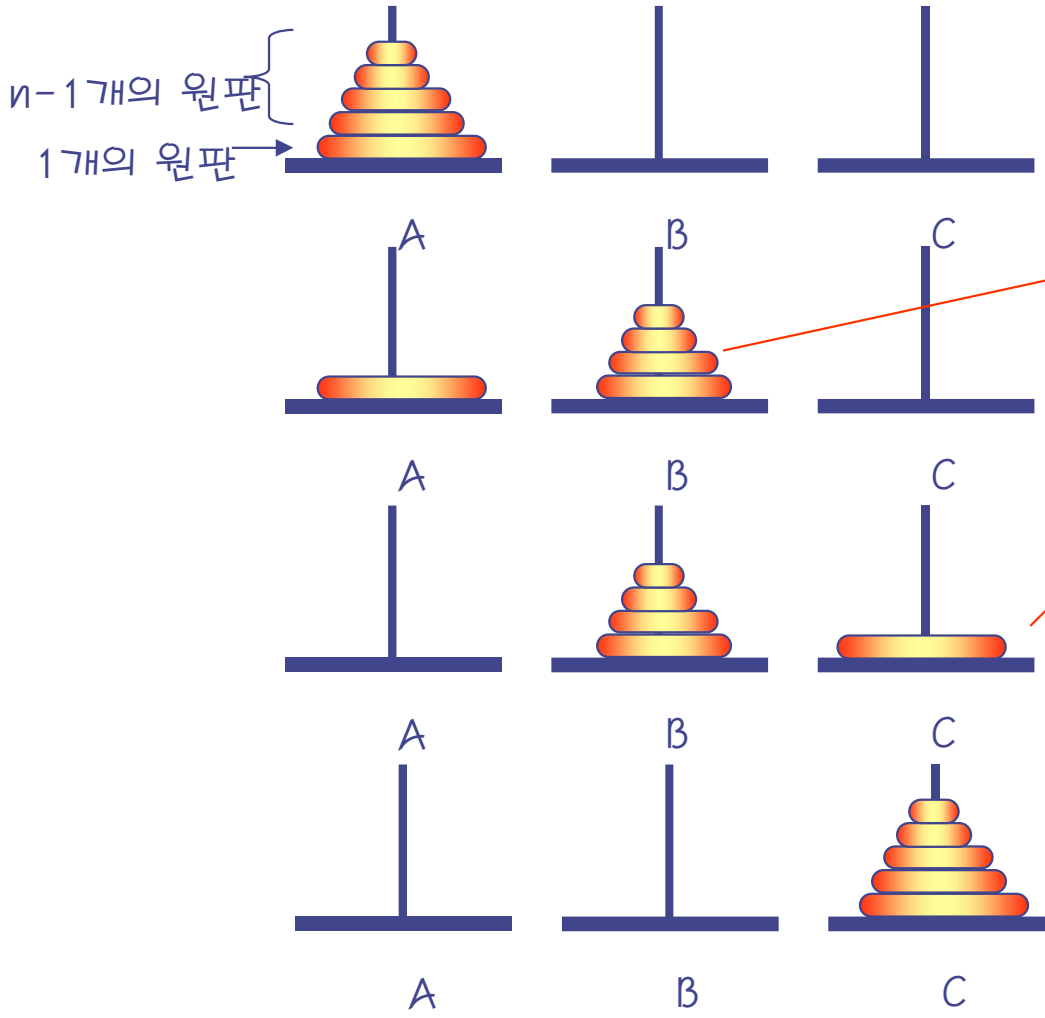
- 문제는 막대 A에 쌓여있는 원판 n개를 막대 C로 옮기는 것이다.
  - 한 번에 하나의 원판만 이동할 수 있다
  - 맨 위에 있는 원판만 이동할 수 있다
  - 크기가 작은 원판 위에 큰 원판이 쌓일 수 없다.
  - 중간의 막대를 임시적으로 이용할 수 있으나 앞의 조건들을 지켜야 한다.



n=3인 경우의 해답



일반적인 경우에는?



A의 n-1개 원판을 B로 옮긴다

A의 가장 큰 원판을 C로 옮긴다.

B에 쌓여있는 n-1개의 원판을 C로 옮긴다.

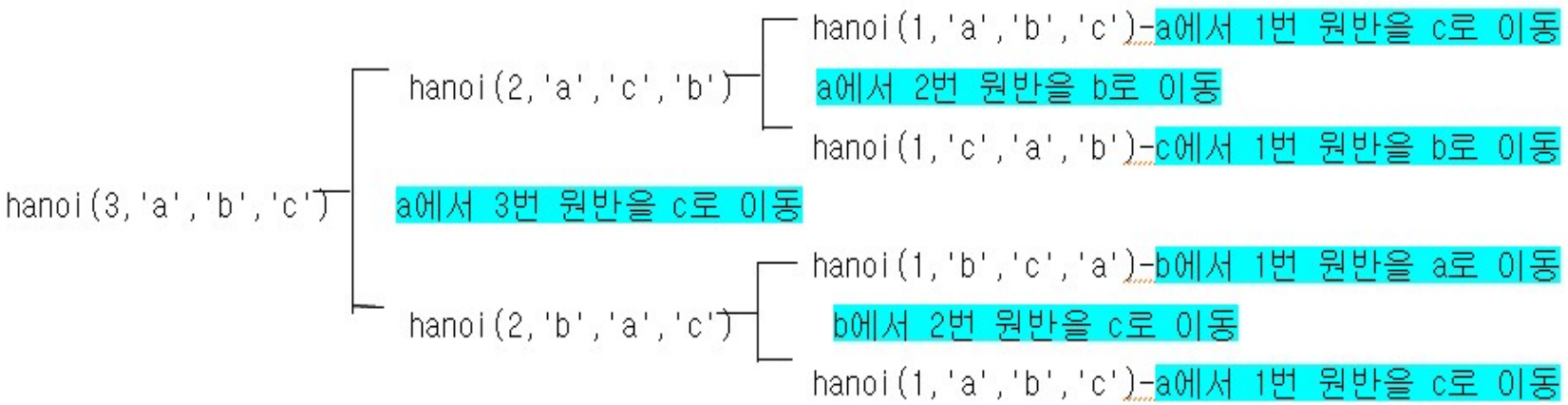
하노이 탑

일반화하여 n개의 원반을 이동하는 방법을 정리해보자.

❶ A의 원반 n-1개를 B으로 옮긴다.

❷ A의 원반 n을 C으로 옮긴다.

❸ B의 원반 n-1을 C으로 옮긴다.

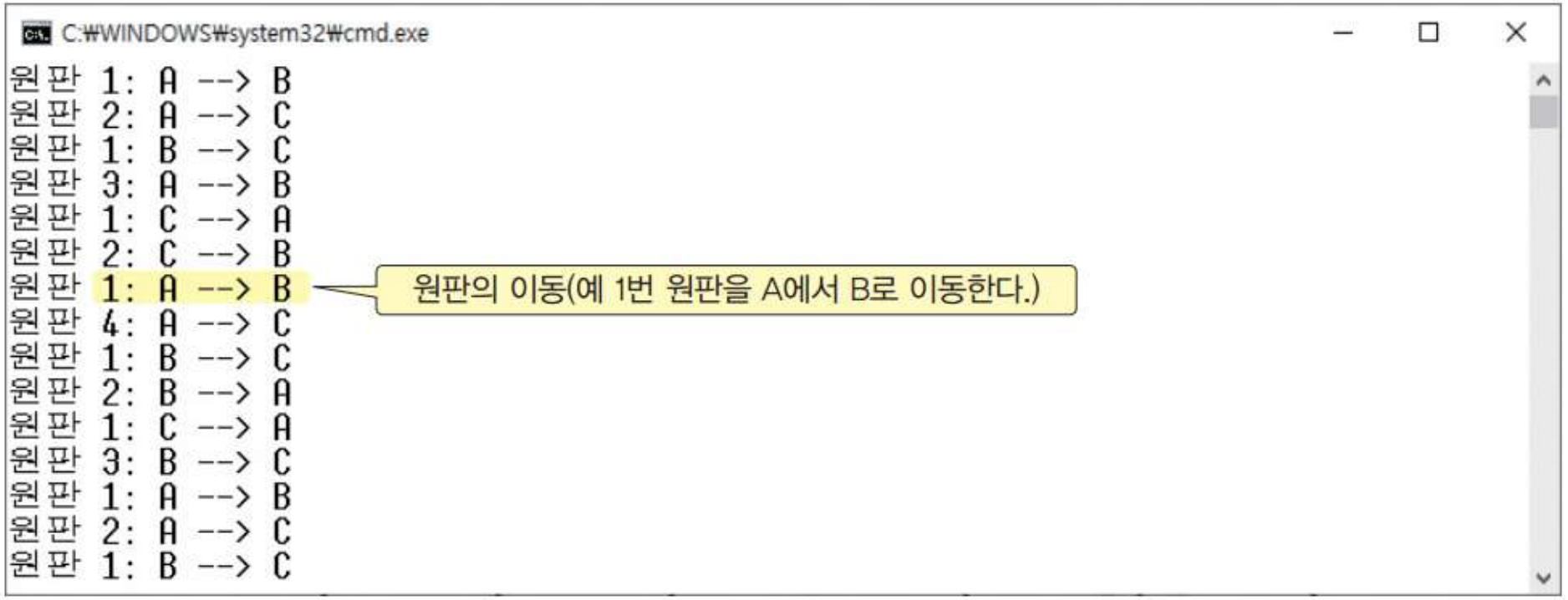


구현

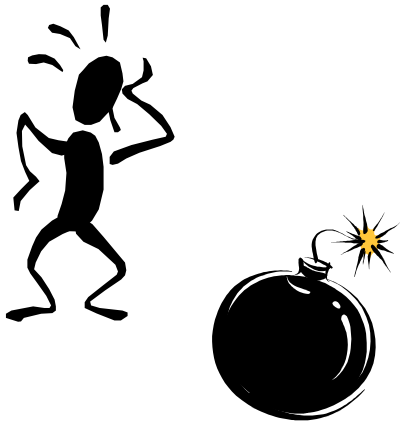
```
def hanoi_tower(n, fr, tmp, to) :                # Hanoi Tower 순환 함수

    if (n == 1) :                                # 종료 조건
        print("원판 1: %s --> %s" % (fr, to))    # 가장 작은 원판을 옮김
    else :
        hanoi_tower(n - 1, fr, to, tmp)          # n-1개를  to를 이용해 tmp로
        print("원판 %d: %s --> %s" % (n,fr,to))  # 하나의 원판을 옮김
        hanoi_tower(n - 1, tmp, fr, to)          # n-1개를  fr을 이용해 to로
```

```
hanoi_tower(4, 'A', 'B', 'C')                    # 4개의 원판이 있는 경우
```

하노이탑(n=4) 실행 결과


C:₩WINDOWS₩system32₩cmd.exe
원판 1: A --> B
원판 2: A --> C
원판 1: B --> C
원판 3: A --> B
원판 1: C --> A
원판 2: C --> B
원판 1: A --> B
원판 4: A --> C
원판 1: B --> C
원판 2: B --> A
원판 1: C --> A
원판 3: B --> C
원판 1: A --> B
원판 2: A --> C
원판 1: B --> C
원판의 이동(예 1번 원판을 A에서 B로 이동한다.)

나쁜 재귀



- 잘못 설계된 재귀
  - **베이스 케이스:** 없음
    - **예:** sum1
  - **재귀 케이스:** 도달 불능 – 즉, 베이스 케이스를 향해 재귀하지 않음
    - **예:** sum2
- 나쁜 재귀 사용의 영향
  - 부정확한 결과
  - 미정지(nontermination)
  - 저장을 위한 기억장소 고갈

```
Alg sum1(n)
1. return n + sum1(n − 1)

Alg sum2(n)
1. if (n = 1)                    {base case}
       return 1
   else                          {recursion}
       return n + sum2(n + 1)
```

01 재귀 호출의 연습

우주선 발사 카운트다운

- 우주선 발사를 위해 카운트하는 코드

카운트다운을 재귀 호출로 구현

```
1  def countDown(n) :
2      if n == 0 :
3          print('발사!!')
4      else :
5          print(n)
6          countDown(n-1)
7
8  countDown(5)
```

실행 결과

```
5
4
3
2
1
발사!!
```

02 재귀 호출의 연습

별 모양 출력하기

- 입력한 숫자만큼 차례대로 별 모양을 출력하는 코드

별 모양 출력을 재귀 호출로 구현

```
1  def printStar(n) :
2      if n > 0 :
3          printStar(n-1)
4          print('★' * n)
5
6  printStar(5)
```

실행 결과

```
★
★★
★★★
★★★★
★★★★★
```

응용예제 01 별 모양 출력하기

별 모양 출력하기

- 입력한 숫자의 크기만큼 차례대로 별 모양을 출력하는 코드를 작성하시오

실행결과

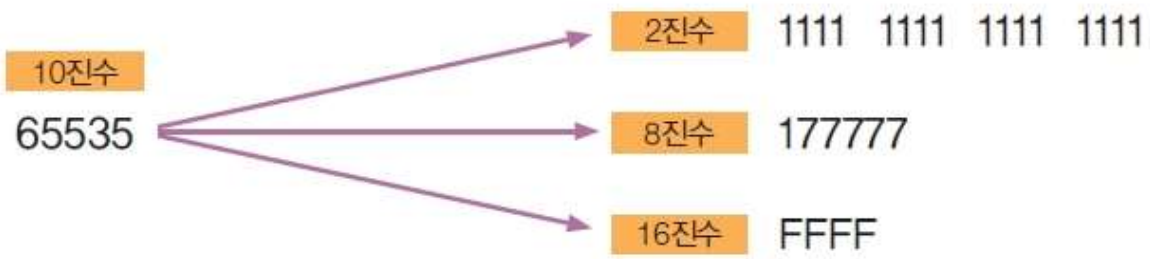
응용예제 01 별 모양 출력하기(정답)

```
def printStar(n) :
        if  ①: n>0
                print  ② '★' * n
                printStar  ③  n-1
printStar(5)
```

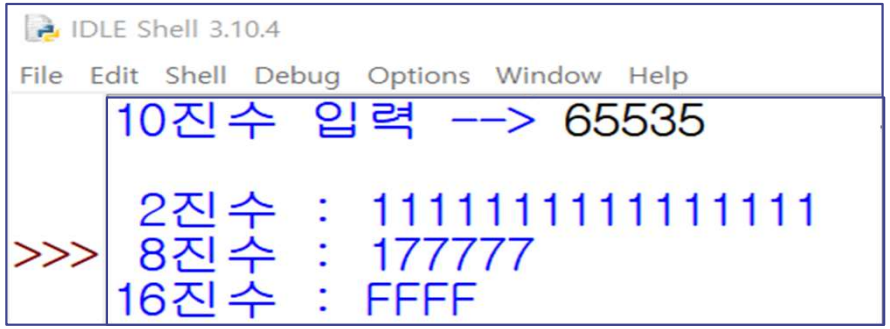
응용예제 02 진수 변환하기

문제

10진수 정수를 입력하면, 2진수/8진수/16진수로 변환되어 출력되는 프로그램을 재귀 함수를 이용하여 작성한다.



실행결과



응용예제 02 진수 변환하기(정답)

```
def notation(base, n):
    if n < 0:
        raise ValueError('Number n must n >=0')
    if n < base :
        print(numberChar[n], end = '')
    else :
        notation(base, n//base)
        print(numberChar[n%base], end = '')

numberChar=['0','1','2','3','4','5','6','7','8',’9’, 'A','B','C','D','E','F']
number = int(input('10진수 입력 --> '))
print('₩n 2진수 : ', end = '')
notation(2, number)
print('₩n 8진수 : ', end = '')
notation(8, number)
print('₩n16진수 : ', end = '')
notation(16, number)
```

응용예제 02 진수 변환하기-비재귀(정답)

```
def notation(base, n):
    if n < 0:
        raise ValueError('Number n must n >=0’)
    digits  = ‘’
    if n == 0:
        digits = numberChar[n]
    else :
        while n > 0 :
            n, m = divmod(n, base)
            digits = numberChar[m] + digits
    print(digits)

numberChar=['0','1','2','3','4','5','6','7','8','9’, 'A','B','C','D','E','F']
number = int(input('10진수 입력 --> '))
print('₩n 2진수 : ', end = '')
notation(2, number)
print('8진수 : ', end = '')
notation(8, number)
print('16진수 : ', end = '')
notation(16, number)
```

Q & A