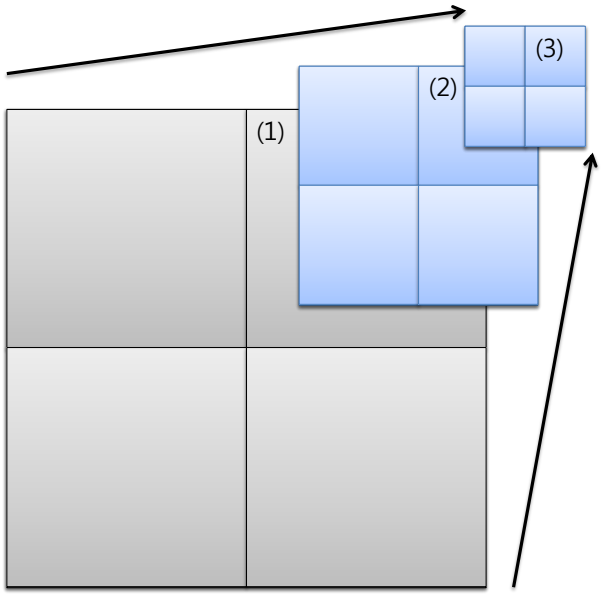
재귀 호출

목차

1. 재귀호출의 개념
2. 재귀호출과 반복호출

1. 재귀호출의 개념 (1/4)

- 재귀 호출(Recursion)
  - 분할 정복(Divide and Conquer)
    - 문제 해결 방식



- 같은 모양, 크기가 작도록 분할
  ≒ 자기 자신을 호출
- 종료 조건(Terminate Condition)

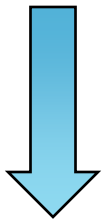
※ 알고리즘의 특성

1) **입력**(input)
2) **출력**(output)
3) **명백성**(definiteness)
4) **유한성**(finiteness)
5) **유효성**(effectiveness)

1. 재귀호출의 개념 (2/4)

- 재귀 호출의 예 (1/2)
  - 팩토리얼 계산

$$n! = \begin{cases} 1 & if\ n = 1 \\ n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 & if\ n \geq 1 \end{cases}$$

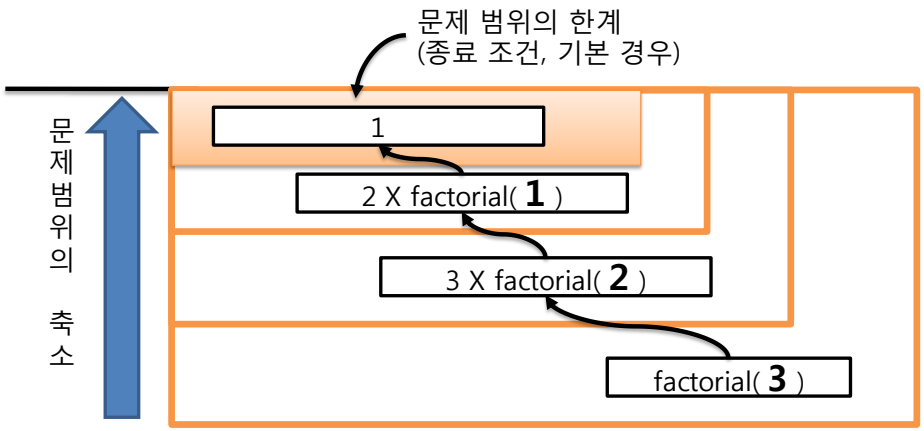


$$n! = \begin{cases} 1 & if\ n = 1 \\ n \times (n-1)! & if\ n \geq 1 \end{cases}$$

1. 재귀호출의 개념 (3/4)

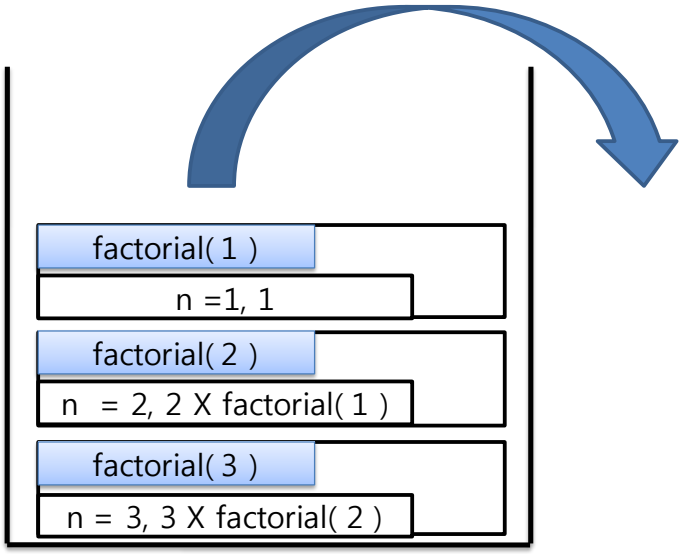
- 재귀 호출의 예 (2/2)
  - 팩토리얼 계산

```
03:  int factorial(int n) {
04:        int ret = 0;
05:
06:        if (n <= 1) {
07:              ret = 1;
08:        }
09:        else {
10:              ret = n * factorial(n - 1);
11:        }
12:
13:        return ret;
14:  }
```



1. 재귀호출의 개념 (4/4)

- 재귀 호출의 내부적 구현
  - 운영체제(OS)에서는 스택(Stack)을 이용하여 재귀 호출을 수행



  - 활성 레코드(Activation Record)
    - 함수를 호출할 때 함수에서 사용되는 모든 지역 변수(Local Variable)와 전달된 인자(Parameter)를 저장하는 공간
    - 활성 레코드를 팝하여 처리하는 과정은 문맥 변경(Context Switch)이 필요

2. 재귀 호출과 반복 호출 (1/5)

- 장점과 단점
  - 재귀 호출
    - 장점: 간결성, 명확성
    - 단점: (문맥변경 등으로) 시간 증가, 메모리 필요
  - 반복 호출
    - 장점: 성능
    - 단점: 구현상의 어려움

- 재귀 호출을 반복 호출로 변경
  - 꼬리 재귀 호출(Tail Recursion)
    - 예) factorial() ⇔ factorial_iter()

2. 재귀 호출과 반복 호출 (2/5)

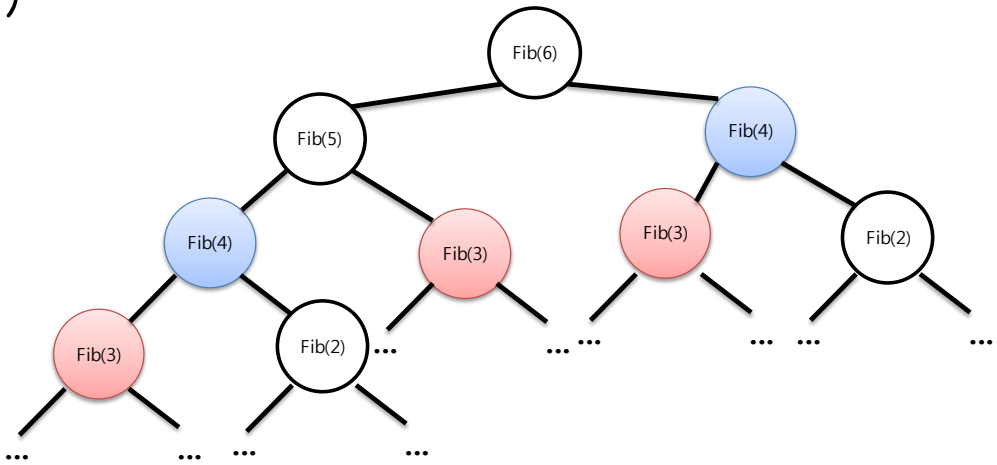
- 반복 호출의 성능 ≫ 재귀 호출의 성능
  - 피보나치 수열

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

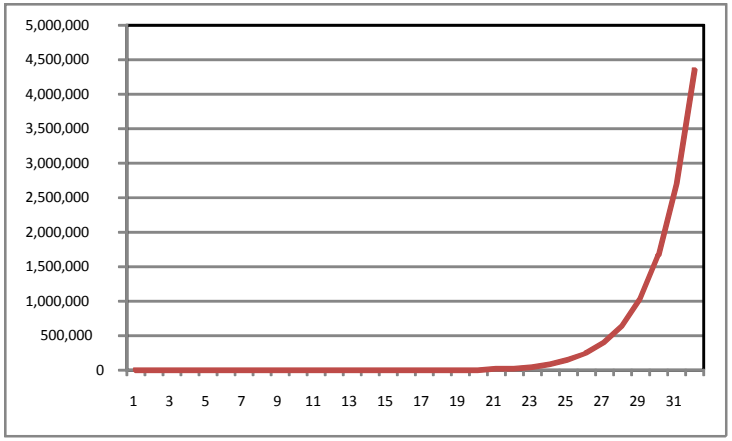
    - 함수 int fib(int n)

2. 재귀 호출과 반복 호출 (3/5)

- 함수의 호출 회수
  - 시간 복잡도: $O(2^n)$

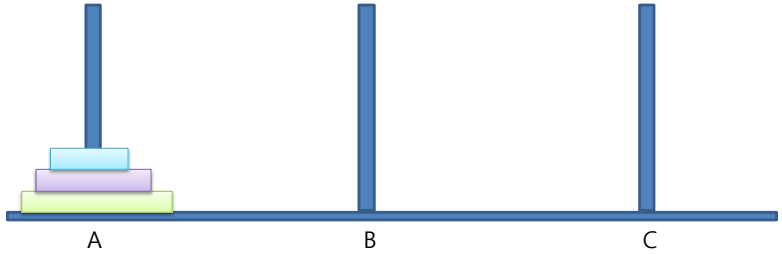


- 반복 호출
  - 함수 int fib_iter(int n)
  - 시간 복잡도: O(n)



2. 재귀 호출과 반복 호출 (4/5)

- 재귀 호출의 장점 (반복 호출의 단점)
  - 하노이의 탑



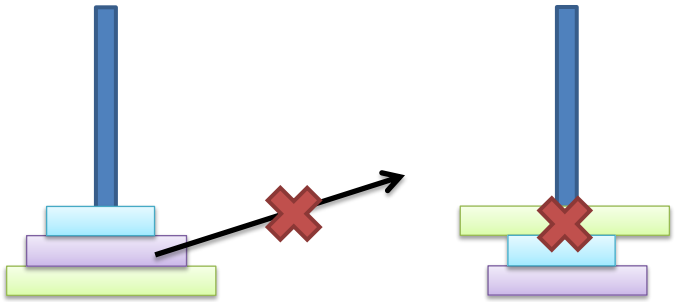
  - 3가지 제약 사항

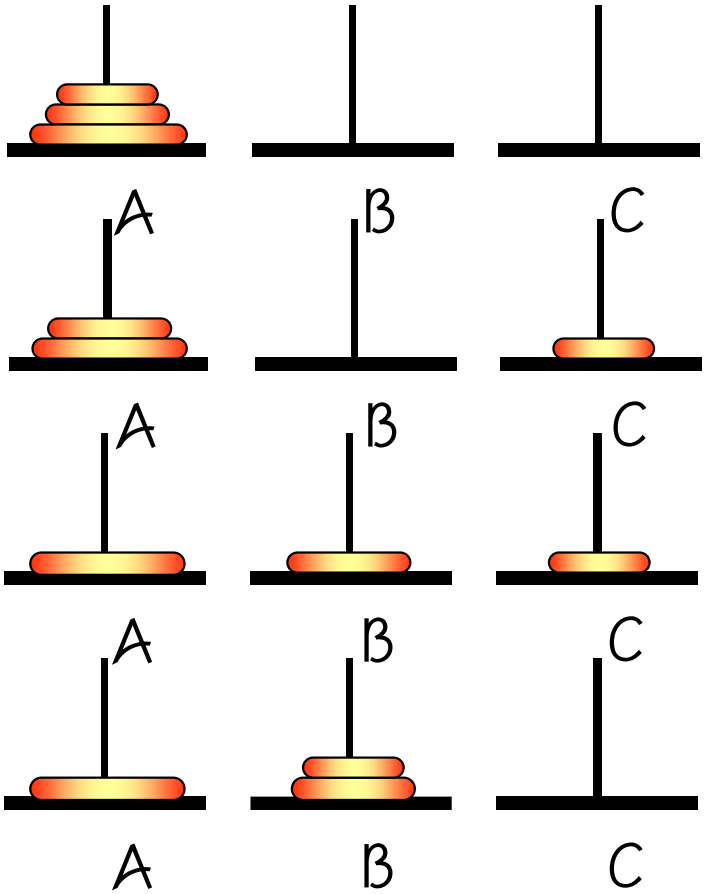
    1) 한 번에 하나의 원판만 이동할 수 있다.

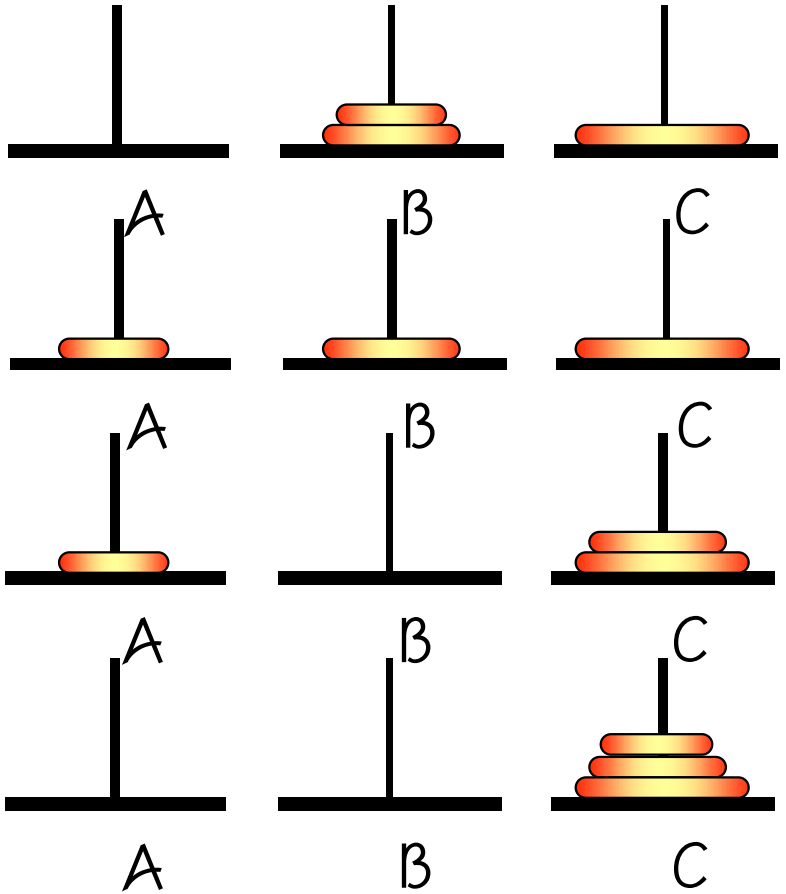
    2) 맨 위에 있는 원판만 이동할 수 있다.

    3) 크기가 작은 원판 위에 큰 원판이 있을 수 없다.
    (크기가 큰 원판 위에만 작은 원판을 놓을 수 있다.)

    4) 중간 막대를 이용할 수 있으나, 앞의 3가지 조건을 만족해야 한다.

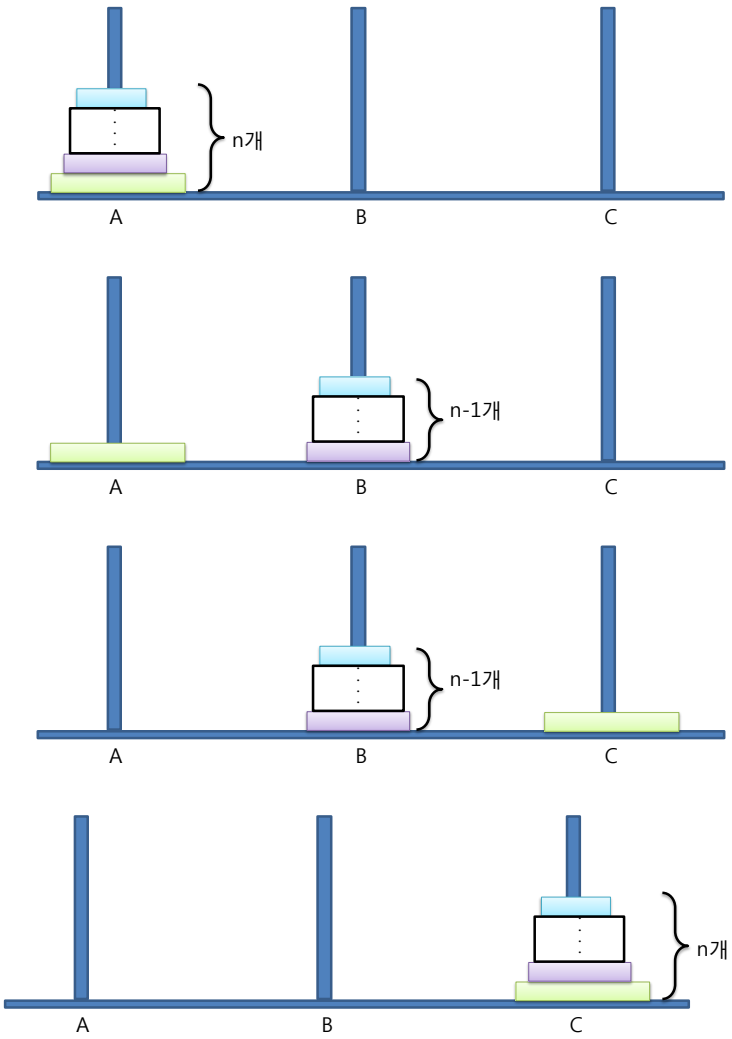



A
B
C
A
B
C
A
B
C
A
B
C
A
B
C


A
B
C
A
B
C
A
B
C
A
B
C
A
B
C

2. 재귀 호출과 반복 호출 (5/5)

- 재귀적 접근 방법




```
void hanoi_tower(int n, char from, char to, char temp) {
      if (n == 1) {
            from에서 to로 원판을 옮긴다
      }
      else {
            Step-1) from의 맨 밑에 원판을 제외한
                        나머지 원판들을 temp로 옮긴다
            Step-2) from에 있는 한 개의 원판을 to로 옮긴다
            Step-3) temp의 원판들을 to로 옮긴다
      }
}
```

```
void hanoi_tower(int n, char from, char to, char temp) {
      if (n == 1) {
            from에서 to로 원판을 옮긴다
      }
      else {
            hanoi_tower(n -1, from, temp, to)
            from에 있는 한 개의 원판을 to로 옮긴다
            hanoi_tower(n -1, temp, to, from)
      }
}
```

이번 장에서는

- 재귀호출의 개념

- 재귀호출과 반복호출