Ch 2 순환 (Recursion, 재귀)

: 함수를 정의할 때 자기 자신을 인용하는 형태로서, 큰 size의 문제를 더 작은 size의 문제(들)로 분해하여 해결하는 방식에 이용

ex) $f(n) = \begin{cases} 1 & \text{if } n=0 \leftarrow \text{terminal (or initial condition)} \\ n * f(n-1) & \text{if } n \ge 1 \leftarrow \text{recurrence relation} \end{cases}$

$f(n) = n!$

기본형: A (···)

```
{
    if (   ) ret  ← condition check to avoid ∞ - loop
    ⋮
    A (   )
    ⋮
}
```

모든 computable func ⟺ recursive func로 구현 가능

ex) ① $add(n, k) = \begin{cases} k & \text{if } n=0 \\ add(n-1, k)+1 & \text{if } n \ge 0 \end{cases}$

$= n+k$

② $sub(n, k) = \begin{cases} -k & \text{if } n=0 \\ sub(n-1, k)+1 & \text{if } n \ge 0 \end{cases}$

$= n-k$

③ $mul(n, k) = \begin{cases} 0 & \text{if } n=0 \\ mul(n-1, k)+k & \text{if } n \ge 0 \end{cases}$

$= n * k$

(or $add(mul(n-1, k), k)$)

④ $div(n, k) = \begin{cases} 0 & \text{if } n<k \\ div(n-k, k)+1 & \text{if } n \ge k \end{cases}$

$= n/k$의 몫

(or $div(sub(n, k), k)+1$)

⑤ $res(n, k) = \begin{cases} n & \text{if } n<k \\ res(n-k, k) & \text{if } n \ge k \end{cases}$

$= n/k$의 나머지

$= n \% k$

순환함수의 장단점

장점 : ① 문제 해결이 쉽다 (← 분할정복법 (Divide & Conquer))
② 알고리즘 표현 및 이해가 쉽다.

단점 : 느리다 (중복 계산 때문)
↳ 동일한 비순환 알고리즘으로 변환하여 실행

$$n! = \begin{cases} 1 & (n=0) \\ n * (n-1)! \end{cases} \qquad n! = 1 \times 2 \times \cdots \times n$$

$f(n) = n^{th}$ 원소 in Fibonacci Sequence

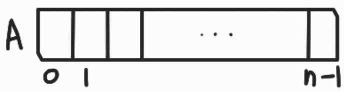
$= 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 \ldots$

$$= \begin{cases} n & \text{if } n \le 1 \\ f(n-1) + f(n-2) & \text{if } n \ge 2 \end{cases}$$

* Divide & Conquer (분할정복법)

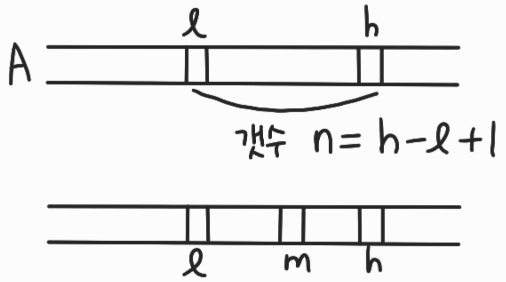
단계
1. divide (분할) → 동일한 유형, 작은 크기의 문제(들)로 분할
2. conquer (정복) → 분할된 문제들을 각각 해결 (순환 호출)
3. combine (병합) → 작은 결과들을 통합하여 원래 문제의 해답으로 정리

ex) Merge sort



MS(0, n-1)

```
MS (int l, int h)
{ if (h ≤ l) ret        // n≤1
  m = ⌊(l+h)/2⌋
  MS(l, m)
  MS(m+1, h)
  merge(l, m, h)  ← 병합
}
```



1 3 5 7 9	2 4 6 8 10

⇒

1 2 3 4 5	6 7 8 9 10

분석

comparison

$M(n)$ = n개를 merge sort 하는 데 걸리는 시간 (비교 횟수)

$$= \begin{cases} 0 & \text{if } n \le 1 \\ 2 \cdot M(\frac{n}{2}) + n & \text{if } n \ge 2 \end{cases}$$

$$\begin{aligned} M(n) &= 2M(\tfrac{n}{2}) + n \\ &= 2[2M(\tfrac{n}{2^2}) + \tfrac{n}{2}] + n \\ &= 2^2 M(\tfrac{n}{2^2}) + n + n \\ &= 2^2[2M(\tfrac{n}{2^3}) + \tfrac{n}{2^2}] + 2n \\ &= 2^3 M(\tfrac{n}{2^3}) + 3n \\ &\ \vdots \\ &= 2^k M(\tfrac{n}{2^k}) + k \cdot n \end{aligned}$$

$\frac{n}{2^k} = 1$, $2^k M(1) = 0$

merge sort는 최선

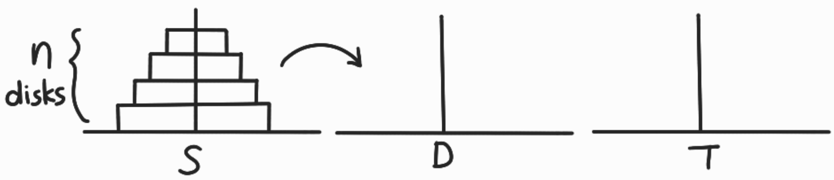
sorting algorithm $= O(n \lg n)$

$f = O(g) \Leftrightarrow f \le g$

$$= kn \qquad \leftarrow n = 2^k \rightarrow k = \lg n$$

$$= n \cdot \lg n = O(n \lg n)$$

ex 2) Hanoi Tower Puzzle



① 한 번에 1 disk만

② 언제나 큰 것이 아래쪽에

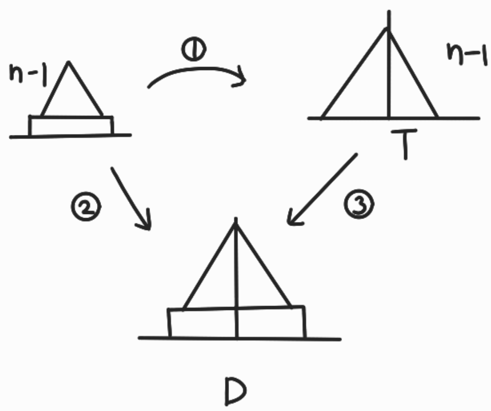
③ 최소의 움직임만으로

$n=2$는 쉽지만 n이 클 때는?

→ 상위 n−1개를 한 묶음으로 보고 두 개의 경우를 적용

HT(n, S, D, T) : S에 있는 n개 disks를 D로 옮기기

```
{   if (n ≤ 0) ret
        HT(n-1, S, T, D)
        S → D
        HT(n-1, T, D, S)
}
```



분석

$M(n)$ = total # of disk moves for n disks

$$= \begin{cases} 0 & \text{if } n \le 0 \\ 2M(n-1)+1 & \text{if } n \ge 1 \end{cases}$$

$$\begin{aligned}
M(n) &= 2M(n-1)+1 \\
&= 2[2M(n-2)+1]+1 \\
&= 2^2 M(n-2)+2+1 \\
&= 2^2[2M(n-3)+1]+2+1 \\
&= 2^3 M(n-3)+2^2+2+1 \\
&= 2^3[2M(n-4)+1]+2^2+2+1 \\
&= 2^4 M(n-4)+2^3+2^2+2+1 \\
&\ \ \vdots \\
&= \underbrace{2^k \underbrace{M(n-k)}_{0}}_{0} + 2^{k-1}+2^{k-2}+\cdots+2^1+2^0 \\
&= 2^{n-1}+2^{n-2}+\cdots+2^1+2^0 \quad (n=k)
\end{aligned}$$

$a.r.n = \frac{a(1-r^n)}{1-r}$

$a=1,\ r=2,\ n=n$

$= 2^n - 1 = O(2^n)$

$n=0$	→ 0번
1	1
2	3
3	7번
5	31

ex) $_nC_k = \binom{n}{k} = \frac{n!}{(n-k)!\,k!} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$$C(n,k) = \begin{cases} 1 & \text{if } n=0 \\ C(n-1,k)+C(n-1,k-1) & \text{if } n \ge 1 \end{cases}$$

배열에 저장하여 중복계산 방지 ⇒ dynamic programming