깃허브 업로드용

2020년 6월 19일 금요일 오전 12:50

```
def ~~ (C, list):
ex_ i = 0                    → O(1)   ∵ list의 크기와 상관없으므로
    n = len(list)            → O(1)
    while i < n :            → 최소의 경우 : 1번만 실행 → O(1)
                               최악의 경우 : n번 실행 → O(n)
        if ~ :
            ~~
    return i                 → O(1)
```

$$\therefore\ O(1)+O(1)+\begin{cases}O(1)\\O(n)\end{cases}+O(1)$$

$= O(4) \rightarrow O(1)$

$= O(n+3) \rightarrow O(n)$

최악의 경우로 생각함.

① n의 값?

input size를 나타내는 변수 ex. O(1)등으로 써도 무관

② 모든 한 줄 코드는 O(1)?

X. input size와 무관한 경우만 O(1)

Input Size

- O(1) : IS가 소요시간에 영향 X
- O(n) : 대체로 1번 반복이 있고, 반복 횟수가 IS와 비례
  $\frac{n}{2}$번, 3n번 반복도 같은 O(n)
- $O(n^2)$: 이중 반복이 모두 IS에 비례
  → $O(n^3)$도 동일 원리
- O(lg n) : 2로 곱하거나 나누는 경우
  2배 절반

· $O(n \lg n)$: $O(n)$과 $O(\lg n)$의 중첩

cf) $O(n^2) = O(n) \cdot O(n)$

2) 공간

↳ 공간복잡도 : IS에 비례하여 사용하는 메모리 공간

Big-O 표기법 사용

· $O(1)$: 수행 과정에서 사용되는 추가 변수의 공간이 IS와 무관

· $O(n)$: " " IS와 정비례

· $O(n^2)$: " " IS의 제곱

5. 재귀함수 Recursion

↳ 스스로를 호출하는 함수

ex_ Factorial.

◦ 재귀적인 문제 풀이 : 같은 형태의 더 작은 문제를 풀고, 본래 문제 해결

ex_ Factorial → $5! = 1\times2\times3\times4\times5$

↓

$4! = 1\times2\times3\times4$

↓

$3! = 1\times2\times3$

↓

$2! = 1\times2$

→ 항상 case를 나누어줘야 함.

- base case : 문제가 충분히 작아 바로 해결가능한 것
  - $n=0$인 경우 → $0!$
- recursive case : 재귀적으로 부분문제를 풀어야 하는 경우
  - $n>0$인 경우

◦ 반복문과의 관계

재귀로 풀 수 있는 문제는 반복문으로 풀 수 있다.

- 반복문과의 관계

재귀로 풀 수 있는 문제는 반복문으로 풀 수 있다.
그 역도 성립. 그러나 매번 효율적인 방법이 다르다.

ex_ 재귀는 함수 수행 후 돌아갈 곳을 Call Stack에 저장
→ 재귀 ↑ ⇒ Call Stack ↑ ⇒ 과부하
Stack overflow