시간 복잡도(Time-complexity)

1. 알고리즘 복잡도 계산이 필요한 이유

하나의 문제를 푸는 알고리즘은 다양할 수 있음

- 정수의 절대값 구하기
  - 1, -1 >> 1
  - 방법 1 : 정수값을 제곱한 값에 다시 루트를 씌우기
  - 방법 2 : 정수가 음수인지 확인해서, 음수일 때만, -1을 곱하기

다양한 알고리즘 중 어느 알고리즘이 더 좋은지를 분석하기 위해, 복잡도를 정의하고 계산함

2. 알고리즘 복잡도 계산 항목

1. 시간 복잡도 : 알고리즘 실행 속도
2. 공간 복잡도 : 알고리즘이 사용하는 메모리 사이즈

가장 중요한 시간 복잡도를 꼭 이해하고 계산할 수 있어야 함

알고리즘 시간 복잡도의 주요 요소

→ **반복문!**

입력의 크기가 커지면 커질수록 반복문이 알고리즘 수행 시간을 지배함

알고리즘 성능 표기법

- Big O(빅 오) 표기법 : O(N)
  - 알고리즘 **최악**의 실행 시간을 표기
  - **가장 많이/일반적으로 사용함**

    - **아무리 최악의 상황이라도, 이정도의 성능은 보장한다는 의미이기 때문**
- Ω (오메가) 표기법 : Ω(N)
    - 오메가 표기법은 알고리즘 **최상**의 실행 시간을 표기
- Θ(세타) 표기법 : Θ(N)
    - 세타 표기법은 알고리즘 **평균** 실행시간을 표기

시간 복잡도 계산은 반복문이 핵심 요소임을 인지하고,계산 표기는 최상, 평균, 최악 중, 최악의 시간인 Big-O표기법을 중심으로 익히면됨

3. 대문자 O표기법

- 빅 오 표기법, Big-O 표기법 이라고도 부름
- O(입력)
    - 입력 n 에 따라 결정되는 시간 복잡도 함수
    - O(1), O(log n), O(n), O(nlog n), O(n^2), O(2^n), O(n!)등으로 표기함
    - 입력 n 의 크기에 따라 기하급수적으로 시간 복잡도가 늘어날 수 있음
        - O(1) < O(log n) < O(n) < O(nlog n) < O(n^2) < O(2^n) < O(n!)
            - 참고: log n 의 베이스는 2 - log_2 n
- 단순하게 입력 n에 따라, 몇번 실행이 되는지를 계산하면 된다.
    - **표현식에 가장 큰 영향을 미치는 n 의 단위로 표기한다.**
    - n이 1이든 100이든, 1000이든, 10000이든 실행을
        - 무조건 2회(상수회) 실행한다: O(1)

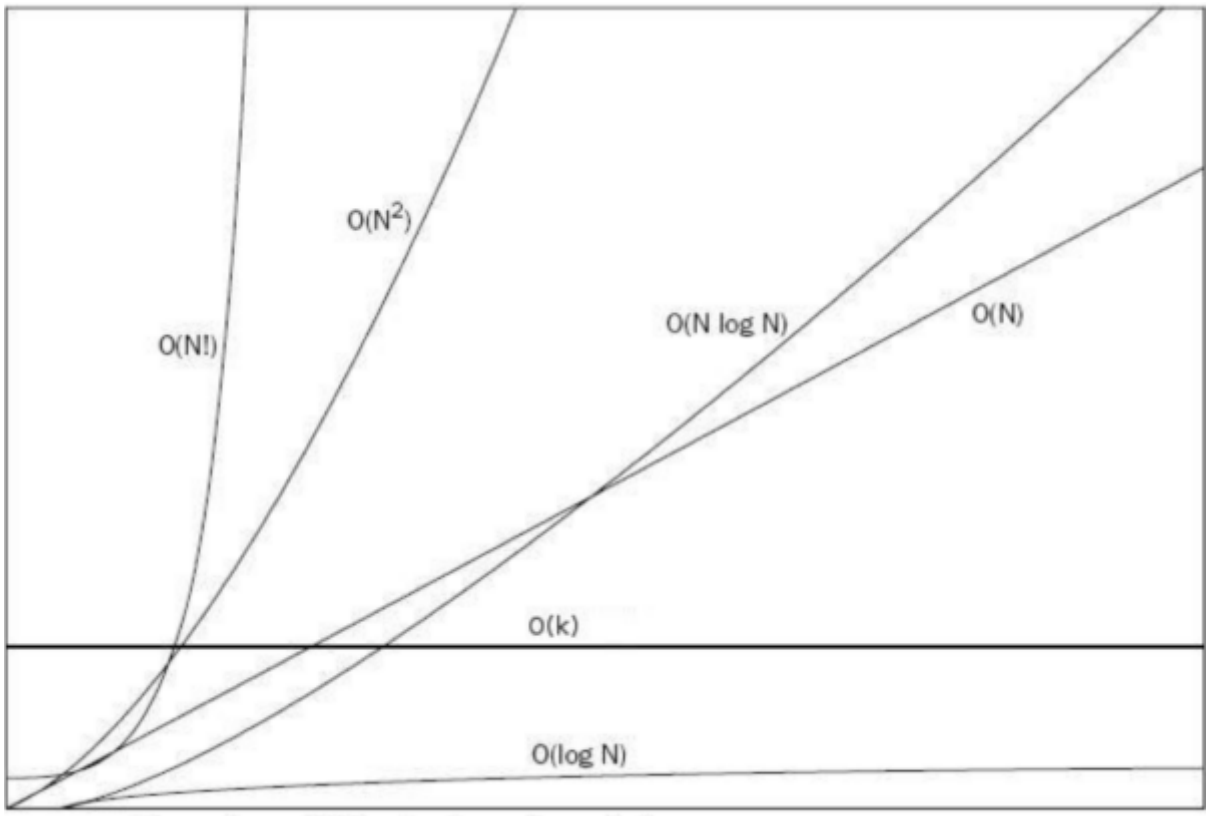
```
if n > 10:
     print(n)
```

        - n에 따라, n번, n + 10 번, 또는 3n + 10 번등 실행한다: O(n)

```
variable = 1
for num in range(3):
    for index in range(n):
         print(index)
```

- n에 따라, n^2번, n^2 + 1000 번, 100n^2 - 100, 또는 300n^2 + 1번등 실행한다: O(n^2)

```
variable = 1
for i in range(300):
    for num in range(n):
        for index in range(n):
             print(index)
```



Comparison of different orders of complexity.

- 빅 오 입력값 표기 방법
  - 예:
    - 만약 시간 복잡도 함수가 2n^2 + 3n 이라면
      - 가장 높은 차수는 2n^2
      - 상수는 실제 큰 영향이 없음

- 결국 빅 오 표기법으로는 O(n^2) (서울부터 부산까지 가는 자동차의 예를 상기)

4. 실제 알고리즘을 예로 각 알고리즘의 시간 복잡도와 빅 오 표기법 알아보기

연습 1 : 1부터 n까지의 합을 구하는 알고리즘 작성해보기

```
def sum(n):
  tot=0
  for i in range(1,n+1):
    tot+=i
  return tot
```

```
sum(100)
>>5050
```

시간 복잡도 구하기

- 1부터 n까지의 합을 구하는 알고리즘1
  - 입력 n에 따라 덧셈을 n번 해야 함(반복문)
  - 시간 복잡도 : n, 빅 오 표기법으로는 **O(n)**

연습2 : 1부터 n까지의 합을 구하는 알고리즘2

```
def sum_2(n):
  return int(n*(n+1)/2)
```

```
sum_2(100)
>>5050
```

시간 복잡도 구하기

- 1부터 n까지의 합을 구하는 알고리즘2

- 입력 n이 어떻든 간에 ,곱셈/덧셈/나눗셈 하면 됨(반복문이 없음)
- 시간 복잡도 : 1, 빅 오 표기법으로는 **O(1)**

어느 알고리즘이 성능이 좋은가?

- 알고리즘1 vs 알고리즘2
  - O(n) vs O(1)

이와 같이, 동일한 문제를 푸는 알고리즘은 다양할 수 있음. 어느 알고리즘이 보다 좋은지를 객관적으로 비교하기 위해, 빅 오 표기법등의 시간복잡도 계산법을 사용함