파이썬으로 배우는 알고리즘 기초

Chap 1. 알고리즘: 효율, 분석, 차수

1.3-1.4 알고리즘의 분석과 차수

1.3 알고리즘의 분석

- 어떤 알고리즘에 대한 두 가지 중요한 질문
  - 정확한가? 모든 입력 사례에 대해서 해답을 찾을 수 있는가?
  - 효율적인가? 입력 크기가 커지면 성능이 어떻게 변화하는가?

- 알고리즘의 분석
  - 정확성 분석: 모든 입력 사례에 대해서 정확한 해답을 찾는다는 것을 증명
  - 효율성 분석: 입력 크기가 커지는 정도에 따라 성능의 변화량을 증명
    - 시간 복잡도(*time complexity*): 시간을 기준으로 알고리즘의 효율성 분석
    - 공간 복잡도(*space complexity*): 공간을 기준으로 알고리즘의 효율성 분석

1.3 알고리즘의 분석

- 알고리즘의 성능 분석
  - 퍼포먼스 측정: 실행 시간을 직접 측정 or 실행 명령의 숫자 세기
    - 컴퓨터의 성능이나 프로그래밍 언어에 따라 달라짐
  - 복잡도 분석: 컴퓨터나 프로그래밍 언어와 무관하게 성능 분석
    - 입력 크기에 따른 단위 연산의 실행 횟수 세기

- 복잡도 분석
  - 입력 크기: 문제가 가진 파라미터, 즉, 입력 사례의 크기
  - 단위 연산: 알고리즘 실행의 기본이 되는 명령어들의 집합

1.3 알고리즘의 분석

- Algorithm 1.2(배열 원소의 합)의 시간 복잡도 분석
  - 단위 연산: 리스트의 원소를 result에 더하는 명령
  - 입력 크기: 리스트 S의 원소 개수(n)
  - for 문장은 항상 n번 실행하므로 다음과 같이 표현
    - 시간 복잡도: $T(n) = n$

```
def sum (S):
    n = len(S)
    result = 0
    for i in range(n):
        result += S[i]
    return result
```

1.3 알고리즘의 분석

- Algorithm 1.3(교환 정렬)의 시간 복잡도 분석
  - 단위 연산: S[i]와 S[j]의 비교
  - 입력 크기: 정렬할 리스트 S의 원소 개수(n)
  - for-j 루프는 i에 따라 $n-1$번에서 1번까지 실행하므로 다음과 같이 계산
  - 시간 복잡도: $T(n) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2}$

```
def exchange (S):
    n = len(S)
    for i in range(n - 1):
        for j in range(i + 1, n):
            if (S[i] > S[j]):
                S[i], S[j] = S[j], S[i] # swap
```

1.3 알고리즘의 분석

- Algorithm 1.4(행렬 곱셈)의 시간 복잡도 분석
  - 단위 연산: 가장 안쪽 for 루프에 있는 곱셈
  - 입력 크기: 행과 열의 개수(n)
  - 3중 for 루프가 항상 n번 실행하므로 다음과 같이 계산
  - 시간 복잡도: $T(n) = n \times n \times n = n^3$

```
def matrixmult (n, A, B):
    C = [[0] * n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

1.3 알고리즘의 분석

- 단위 연산의 실행 횟수는 항상 일정한가?
  - Algorithm 1.2, 1.3, 1.4의 경우: 항상 일정
  - Algorithm 1.1(순차 탐색)의 경우: 입력 사례에 따라 다름
    - $S = [10, 7, 11, 5, 13, 8], x = 3$
    - $S = [10, 7, 11, 5, 13, 8], x = 5$
    - $S = [10, 7, 11, 5, 13, 8], x = 10$

- 입력 사례에 따른 시간 복잡도 분석
  - 일정 시간 복잡도: 입력 사례에 따라 달라지지 않는 경우
  - 최악, 최적, 평균 시간 복잡도 분석: 입력 사례에 따라 달라지는 경우

1.3 알고리즘의 분석

- Algorithm 1.1(순차 탐색)의 시간 복잡도 분석
  - 단위 연산: 리스트의 원소와 주어진 키 x와의 비교 연산
  - 입력 크기: 리스트 원소의 개수(n)
  - **최악**의 경우는 모두 비교: $W(n) = n$
  - **최적**의 경우는 한 번만 비교: $B(n) = 1$
  - **평균**의 경우: 주어진 키 x가 k번째에 있으면 k번을 비교함
    - 만약 어떤 키 x는 리스트 S에 골고루 분포해 있다고 한다면,

$$A(n) = \sum_{k=1}^{n} \left(k \times \frac{1}{n} \right) = \frac{1}{n} \sum_{k=1}^{n} k = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

1.4 알고리즘의 차수

- 어떤 알고리즘이 (궁극적으로) 더 빠른가?
  - 시간 복잡도: 입력 크기(n)에 대한 단위 연산 횟수의 함수 $f(n)$
  - 시간 복잡도가 $f_1(n) = n$인 알고리즘과 $f_2(n) = n^2$ 인 알고리즘
  - 만약 단위 연산의 실행 시간이 f_2는 t이고, f_1은 $1000t$일 경우
    - f_1의 단위 연산이 f_2의 단위 연산보다 1000배 느리지만
    - 알고리즘의 전체 실행 시간은 f_1이 $n \times 1000t$, f_2는 $n^2 \times t$이므로
    - 부등식 $n^2 \times t > n \times 1000t$이 성립하려면
    - $n > 1000$
    - 즉, n이 1000보다 크면 f_1이 f_2보다 궁극적으로 더 빠르다고 할 수 있다.

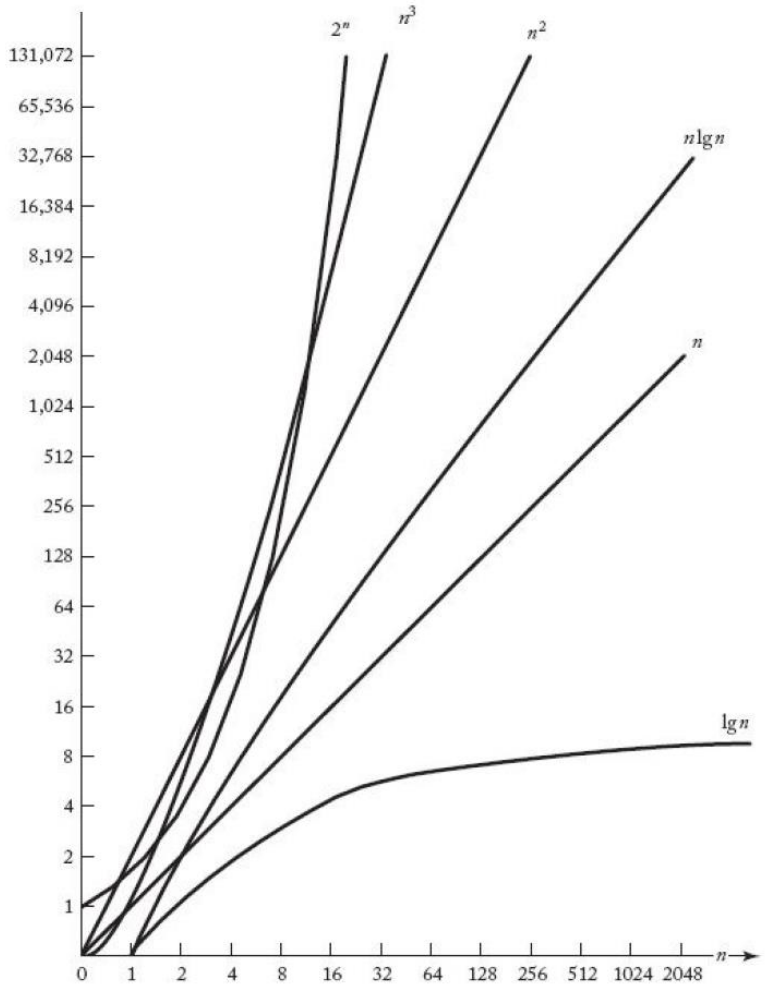
1.4 알고리즘의 차수

- **차수(Order)**: 알고리즘의 궁극적인 **성능 분류**
  - 1차 시간 알고리즘: 시간 복잡도가 1차 함수인 알고리즘
  - 2차 시간 알고리즘: 시간 복잡도가 2차 함수인 알고리즘
  - 근본 원리: 모든 1차 시간 알고리즘은 궁극적으로 2차 시간 알고리즘보다 빠르다.
  - 따라서, **시간 복잡도 함수의 차수**로 **알고리즘의 성능을 분류**할 수 있다.

1.4 알고리즘의 차수

- 자주 사용되는 복잡도 분류

복잡도 함수	분류 이름	대분류
1	상수 시간	다항 시간
$\lg n$	로그 시간	
n	선형 시간	
$n \lg n$	선형-로그 시간	
n^2	2차 시간	
n^3	3차 시간	
2^n	지수 시간	지수 시간
$n!$	팩토리얼 시간	



1.4 알고리즘의 차수

- 알고리즘의 성능을 차수로 분류하는 법
  - 복잡도 함수를 분류할 때 낮은 차수의 항들은 항상 버릴 수 있다.
  - 예를 들어, $an^2 + bn + c$의 함수를 2차 시간 함수(n^2) 로 분류
  - (궁극적으로) 2차 항이 이 함수의 값을 결정하는 데 가장 중요하기 때문

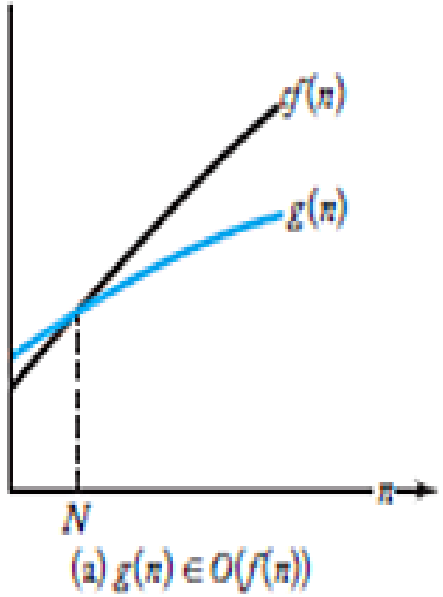
- 점근적 표기법: O, Θ, Ω
  - 빅오(O): 복잡도 함수의 점근적 상한을 표기
  - 오메가(Ω): 복잡도 함수의 점근적 하한을 표기
  - 세타(Θ)=차수: 복잡도 함수의 점근적 상한과 하한을 동시에 만족

1.4 알고리즘의 차수

빅오(O)의 정의: 점근적 상한

For a given complexity function $f(n)$, $O(f(n))$ is the ***set*** of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that, for all $n \geq N$,

$$g(n) \leq c \times f(n)$$



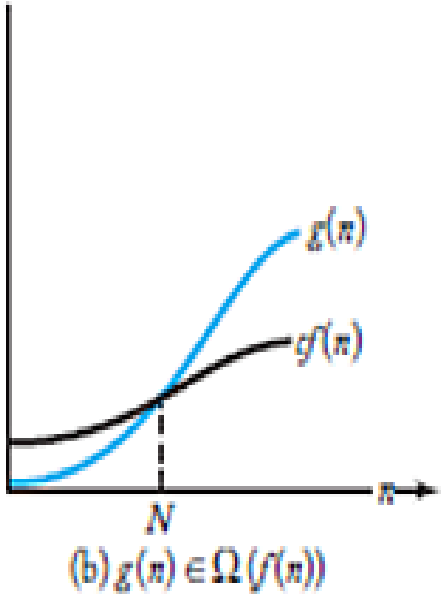
- $g(n) = n^2 + 10n,\ f(n) = n^2$
- $n^2 + 10n \leq 2n^2, n \geq 10$
- $c = 2, N = 10$
- $g(n) \in O(f(n))$
- **그러므로**, $T(n) = n^2 + 10n \in O(n^2)$

1.4 알고리즘의 차수

- **오메가(Ω)의 정의: 점근적 하한**

For a given complexity function $f(n)$, $\Omega(f(n))$ is the ***set*** of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that, for all $n \geq N$,

$$g(n) \geq c \times f(n)$$



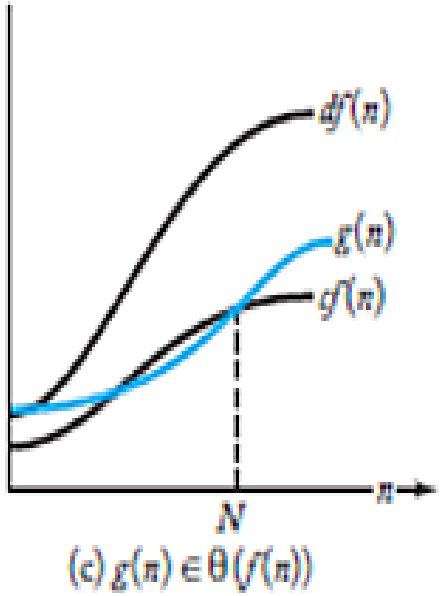
- $g(n) = n^2 + 10n,\ f(n) = n^2$
- $n^2 + 10n \geq n^2, n \geq 0$
- $c = 1, N = 0$
- $g(n) \in \Omega(f(n))$
- **그러므로**, $T(n) = n^2 + 10n \in \Omega(n^2)$

1.4 알고리즘의 차수

- **쎄타(Θ)의 정의: 접근적 상한(O) + 점근적 하한 (Ω)**

For a given complexity function $f(n)$,

$$\Theta(f(n)) = \Omega(f(n)) \cap O(f(n))$$



- $g(n) = n^2 + 10n,\ f(n) = n^2$
- $g(n) \in O(n^2)$
- $g(n) \in \Omega(n^2)$
- **그러므로**, $T(n) = n^2 + 10n \in \Theta(n^2)$

1.4 알고리즘의 차수

- Algorithm 1.3(교환 정렬)의 차수
  - $T(n) = \frac{(n-1)n}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$
  - $T(n) \in O(n^2), T(n) \in \Omega(n^2), T(n) \in \Theta(n^2)$
  - **그러므로, 교환 정렬 알고리즘의 차수는** $\Theta(n^2)$

1.4 알고리즘의 차수

- Algorithm 1.1(순차 탐색)의 차수
  - **최악**의 경우: $W(n) = n \in \Theta(n)$
  - 최적의 경우: $B(n) = 1 \in \Theta(1)$
  - **평균**의 경우: $A(n) = \frac{n+1}{2} \in \Theta(n)$

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t