📌

알고리즘에 대한 점근적 분석

알고리즘

- 작업을 수행하는 절차의 집합
- 가장 중요한 속성- 정확성(correctness), 효율성(efficiency)

🌟정확성과 효율성, 알고리즘에서 가장 중요한 속성

1. 정확성 Correctness

- 주어진 입력에 대해서 모두 처리하고, 올바르게 출력하는 것

2. 효율성 Efficiency

- 알고리즘이 ☑️**메모리를 어느정도 사용**하고, ✅**얼마나 빠른 시간 내**에 결과를 출력하는 지 측정하는 것

- ☑️**시간 복잡도(Time Complexity)** + ✅**공간 복잡도(Space Complexity)** 를 이용하여 효율성을 측정, 평가(알고리즘의 효율성을 평가하는 두 가지 변수!!)

- 시간 복잡도 : 알고리즘이 얼마나 빠르게 결과를 출력하는 지 측정

  🐣 **T(n)** : 입력 크기 n에 대한 시간을 나타내는 함수로, 시간 복잡도를 나타냄!

- 공간 복잡도 : 원하는 결과를 출력하기 위해 필요한 메모리를 측정

🐣 S(n) : 입력 크기 n에 대한 메모리 사용을 나타내는 함수로, 공간 복잡도를 나타냄!

🌟점근적 분석 Asymptotic Analaysis

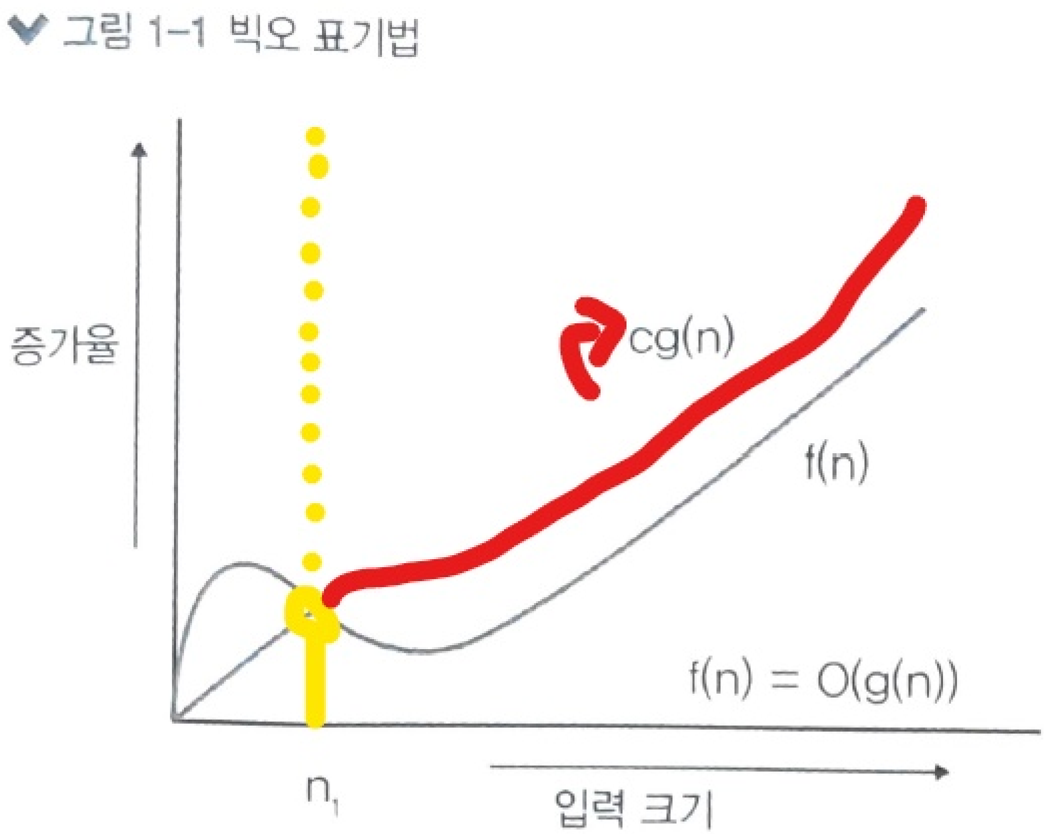
- 데이터 집합이나 프로그래밍 언어와 관계없이 **알고리즘 자체의 효율성을 비교**하기 위해 사용되는 분석 방법으로, **증가차수***에 초점이 맞추어져 있음
- *** 증가차수** : "△(입력 양 증가 정도) ➡️ 알고리즘 수행 시간" 경향성을 나타내는 지표
- **점근적 실행 시간 Asymptotic Running Time** : 입력 양이 증가하는 정도에 따라 변화하는 알고리즘의 수행 시간! 정확한 시간은 아님!

1. 🌟**빅오 표기법 Big-O Notation**

> ∀ $n \geq n_0$에 대해서 조건 $f(n) \leq cg(n)$ ➡️ $f(n) =$ 복잡도 $= O(g(n))$
>
> ↔ f(n)= g(n)의 빅오(Big-O)[f(n): 내가 설정했던 복잡도]
> (단, $n_0 > 0, c > 0$)
>
> (↔즉, 입력크기 n이 커질 수록, 이에 대한 복잡도를 나타내는 f(n) 함수에 대해서 **cg(n)이 상한이 되는 조건**!)

↔ 입력크기 n이 충분히 크면, f(n)의 증가 속도 ≤ cg(n)

- 상한 : 실행 시간이 최악일 경우, 상한 시간과 같거나 빠름



Big O Notation 빅오 표기법

예시)

$$O(n^2) = n^2 + n$$

↔ 이 경우, 양수 n에 대해서

$f(n) = n^2 + n$, $g(n) = n^2$

와 같이 두 함수 f(n), g(n)이 있을 때, c를 6 으로 선택(가정)해보고, 양수값 n에 대해서

$f(n) \leq 6 * g(n)$ 이 되는 경우가 존재하는 지 확인해보자

$n^2 + n$ ≤ $6 * n^2$ ↔ 0 ≤$5n^2 - n$

↔ 0 ≤ $n(5n-1)$

↔ n ≥ 0.2(=1/5) 이면 위에서 언급한 조건이 만족된다!

이 때, 조건을 만족하는 (c, n_0) = (6, 0.2) 를 "witness pair" 로 부를 수 있다!

(reference : http://www.cs.cornell.edu/courses/cs211/2005sp/Lectures/L14-BigO/L14-15-Complexity.4up.pdf)

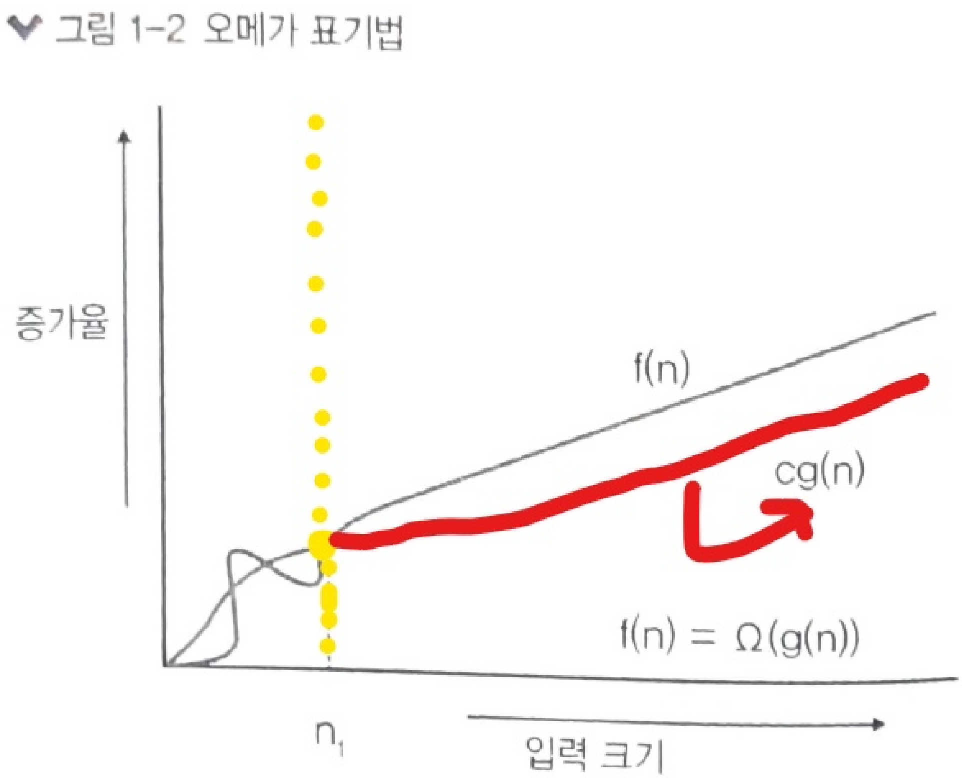
$\therefore f(n) = O(g(n)) = n^2$

2. 오메가 표기법 Omega Notation

> ∀ n ≥ n_0에 대해서 조건 $f(n)$ ≥ $cg(n)$ ➡️ f(n)= Ω(g(n))
>
> ↔ f(n)= g(n)의 Ω(Omega)[f(n): 내가 설정했던 복잡도]
> (단, $n_0 > 0, c > 0$)
>
> (↔즉, 입력크기 n이 커질 수록, 이에 대한 복잡도를 나타내는 f(n) 함수에 대해서 **cg(n)이 하한이 되는 조건**!)

↔ 입력크기 n이 충분히 크면, f(n)의 증가 속도 ≥ cg(n)

- 하한 : 실행 시간이 최악일 경우, 하한 시간과 같거나 느림



Omega Notation-오메가 표기법

예시)

f(n) = n^2+n , g(n) =n^2, c = 1 로 가정하자

n^2+n ≥ n^2

↔ n ≥ 0 이면 f(n)≥g(n)이 성립!

$\therefore f(n) = \Omega(g(n)) = \Omega(n^2)$

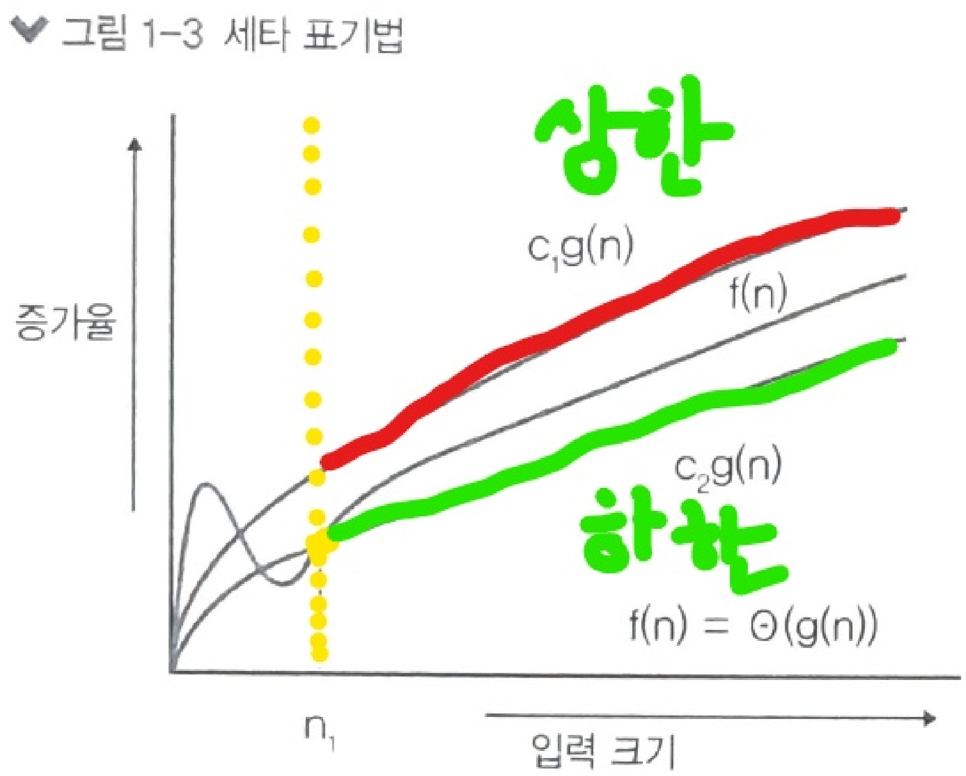
3. 세타 표기법 Theta Notation

> ∀ n ≥ n_0에 대해서 조건 $c_1g(n)$ ≤ $f(n)$ ≤ $c_2g(n)$ ➡️ f(n)=Θ(g(n))

↔ f(n)= g(n)의 Θ(Theta)[f(n): 내가 설정했던 복잡도]
(단, $n_0 > 0, c_1 > 0, c_2 > 0$)

(↔ **g(n) = f(n)에 대해 점근적 근접 한계값* Asymptotically Tight Bound**)

↔ f(n)은 g(n)과 같은 비율로 증가



Theta Notation-세타 표기법

- * 점근적 근접 한계값 Asymptotically Tight Bound

: 점근적 상한과 점근적 하한의 교집합

예시)

$f(n) = n^3 + n^2 + n,\ g(n) = n^3$, $c_1 = 1, c_2 = 3$ 가정

➡️ n^3 ≤ $n^3 + n^2 + n$ ≤ $3n^3$

(1) n^3 ≤ $n^3 + n^2 + n$

↔ $n^2 + n$ ≥ 0

↔ n≥0 인 모든 값에 대해서 성립

(2) $n^3 + n^2 + n$ ≤ $3n^3$

↔ 0 ≤ $2n^3 - n^2 - n$

↔ 0 ≤ $n(2n^2 - n - 1)$

↔ 0 ≤ $n(2n+1)(n-1)$

↔ n≥1 이면 성립

$\therefore\ f(n) = \Theta(g(n)) = \Theta(n^3)$

예제) $f(n) = 2n^2 + n, g(n) = n^2$ 의 관계를 각 표기법에서 살펴보기

(1) 빅오 표기법

c = 4 가정

$2n^2 + n$ ≤ $4n^2$

$2n^2 + n$ ≤ $4n^2$

↔ 0 ≤ $2n^2 - n$

↔ 0 ≤ $n(2n - 1)$

↔ n ≥ 0.5 이면 성립!

$\therefore f(n) = O(g(n)) = n^2$

(2) 오메가 표기법

c =1 가정

$2n^2 + n$ ≥ n^2

↔ $n^2 + n$ ≥ 0

↔ n ≥ 0 이면 성립!

$\therefore f(n) = \Omega(g(n)) = \Omega(n^2)$

(3) 세타 표기법

c_1= ,2 c_2 $=$6가정

2n^2 ≤ $2n^2 + n$ ≤ 6n^2

i) $2n^2$ ≤ $2n^2 + n$

↔ n ≥ 0 이면 성립!

ii) $2n^2 + n$ ≤ $6n^2$

↔ 0 ≤ $4n^2 - n$

↔ 0 ≤ $n(4n - 1)$

↔ n ≥ 0.25 이면 성립!

∴ n ≥ 0.25 면 성립!!

$$\therefore f(n) = \Theta(g(n)) = \Theta(n^2)$$

- 점근적 분석 방법은 완벽하지는 않지만, 알고리즘을 분석하는 가장 좋은 방법이 될 수 있음

(비유) LinkedList와 ArrayList는 각각 중간에서 삽입하느냐, 앞뒤에서 데이터를 삽입하느냐에 따라 성능의 차이가 존재한다

하지만, 데이터가 커지면 LinkedList가 훨씬 유리한 반면, 삽입되는 데이터 양이 작으면 비슷하거나 어느 한쪽이 더 우세하다

이 관점을 끌고 하나의 예시를 살펴보자

$$f(n) = 1000 * n * log(n), g(n) = n^2$$

위와 같은 복잡도 함수에서 상수는 무시되므로, f(n)이 보다 우세하다!

하지만!! n<1000 인 경우에서는 g(n)이 더 빠르게 실행될 수 있다!

즉, 상황에 따라서 상대적으로 적용될 수 있음을 잊지 말자!