컴퓨터 알고리즘

알고리즘 분석

2018. 3. 11.

천 지 영

알고리즘 분석

분석 방법

- Every-case time complexity: $T(n)$
  - Only depending on input size
  - Input values와는 무관하게 결과 값은 항상 일정
- Worst-case time complexity: $W(n)$
  - Depending on input size and input values
  - 기본 연산이 수행되는 횟수가 최대인 경우 선택
- Average-case time complexity: $A(n)$
  - Depending on input size and input values
  - 기본 연산이 수행되는 기대치(평균)
  - Each input에 대해서 probability assignment 가능
  - 일반적으로 최악의 경우보다 계산이 복잡
- Best-case time complexity: $B(n)$
  - Depending on input size and input values
  - 기본 연산이 수행되는 횟수가 최소인 경우 선택

알고리즘 분석

- Add Array Items
  - addition
    - $T(n) = n$

```
number sum (int n, const number S[])
{
  index i;
  number result;

  result = 0;
  for (i = 1; i <= n; i++)
    result = result + S[i];
  return result;
}
```

- Sequential Search
  - 비교연산
    - $W(n) = n$
    - $B(n) = 1$
    - $A(n) = \frac{n+1}{2}$

```
void seqsearch(int n,
               const keytype S[],
               keytype x,
               index& location)
{
  location = 1;
  while (location <= n && S[location] != x)
    location++;
  if (location > n)
    location = 0;
}
```

차수 표기법

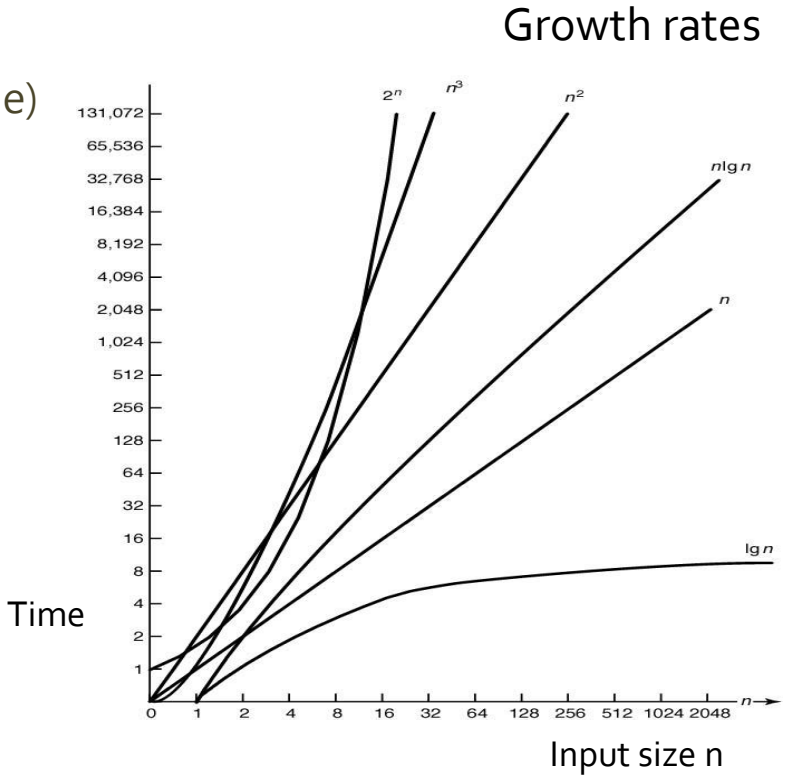
- Let $T(n)$ of algorithm A = $0.01n^2$ and $T(n)$ of algorithm B = $100n$. Which one is better?
  - B is not always better than A
  - 예를 들어, n이 100 이하라면 알고리즘 A가 시간이 적게 걸리고, n이 10000 이상이라면 알고리즘 B가 시간이 적게 걸림
    - $0.01n^2 > 100n \rightarrow n > 10{,}000$
- 어떤 알고리즘이 효율적인지 결정하기 위한 measure가 있어야 하지 않나?
  - ➔ 일반적으로, 입력 크기 n이 매우 커진다고 가정하고 비교함

n	$0.1n^2$	$0.1n^2+n+100$
10	10	120
20	40	160
50	250	400
100	1,000	1,200
1,000	100,000	101,100

  - ➔ 결국 이차항이 이 함수의 값을 지배함

차수 표기법

- 복잡도 부류
  - $\Theta(1)$ 상수 시간 (Constant time)
  - $\Theta(\log n)$ 로그(대수) 시간 (Logarithmic time)
  - $\Theta(n)$ 선형 시간 (Linear time)
  - $\Theta(n\log n)$ 로그 선형 시간 (Log-linear time)
  - $\Theta(n^2)$ 제곱 시간 (Quadratic time)
  - $\Theta(n^3)$ 세제곱 시간 (Cubic time)
  - $\Theta(2^n)$ 지수 시간 (Exponential time)



차수 표기법

- 실행 시간

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 μs*	0.01 μs	0.033 μs	0.10 μs	1.0 μs	1 μs
20	0.004 μs	0.02 μs	0.086 μs	0.40 μs	8.0 μs	1 ms†
30	0.005 μs	0.03 μs	0.147 μs	0.90 μs	27.0 μs	1 s
40	0.005 μs	0.04 μs	0.213 μs	1.60 μs	64.0 μs	18.3 min
50	0.006 μs	0.05 μs	0.282 μs	2.50 μs	125.0 μs	13 days
10^2	0.007 μs	0.10 μs	0.664 μs	10.00 μs	1.0 ms	4×10^{13} years
10^3	0.010 μs	1.00 μs	9.966 μs	1.00 ms	1.0 s	
10^4	0.013 μs	10.00 μs	130.000 μs	100.00 ms	16.7 min	
10^5	0.017 μs	0.10 ms	1.670 ms	10.00 s	11.6 days	
10^6	0.020 μs	1.00 ms	19.930 ms	16.70 min	31.7 years	
10^7	0.023 μs	0.01 s	2.660 s	1.16 days	31,709 years	
10^8	0.027 μs	0.10 s	2.660 s	115.70 days	3.17×10^7 years	
10^9	0.030 μs	1.00 s	29.900 s	31.70 years		

*1 μs = 10^{-6} second.

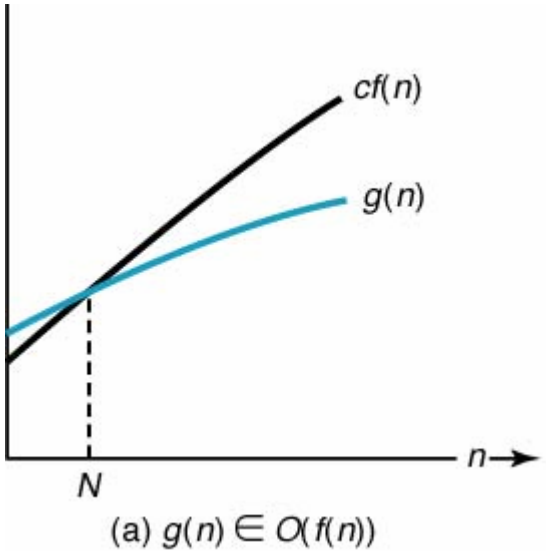
†1 ms = 10^{-3} second.

차수 표기법: O(Big-Oh)-표기

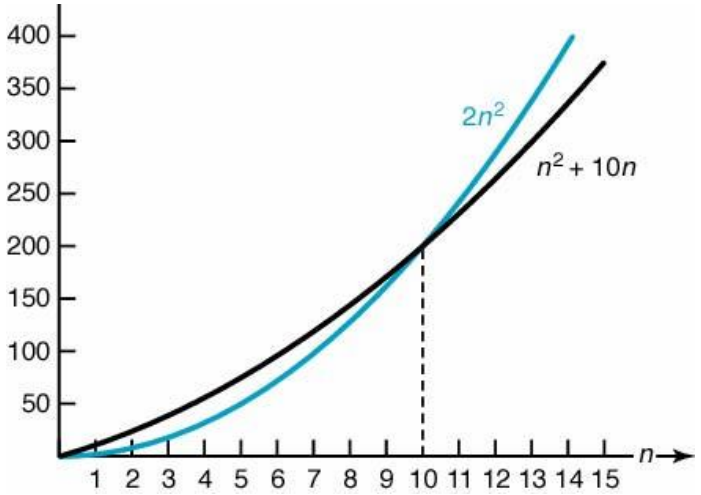
- O(Big-Oh)-표기
  - 정의: $n \geq n_0$인 모든 n에 대해서 $g(n) \leq c \times f(n)$을 만족하는 상수 c와 n_0가 존재하면, $g(n) \in O(f(n))$이다.
  - $g(n)$이 비록 $cf(n)$보다 위에서 시작하지만 어느 시점이 되면 $cf(n)$보다 밑으로 내려간다.

$$n^2 + 10n \leq 2n^2, n \geq 10$$

$$n^2 + 10n \in O(n^2)$$



(a) $g(n) \in O(f(n))$



차수 표기법: O(Big-Oh)-표기

- $O(f(n))$
  - $g(n)$이 $O(n^2)$이고 $g(n)$이 어떤 알고리즘의 시간 복잡도이면 그 알고리즘의 실행시간은 이차함수와 같거나 좋다는 것을 의미한다.
  - 점근적 증가율이 $f(n)$을 넘지 않는 모든 함수들의 집합, 함수의 점근적 상한
  - $5n^2 + 4n = O(n^2),\ 7n = O(n^2),\ 2n\log n + 3n = O(n^2)$
    - $g(n) \in O(f(n))$을 관행적으로 $g(n) = O(f(n))$이라고 쓴다.

- Describe $T(n)$ as tight as possible
  - $n\log n + 5n = O(n\log n)$ 인데 굳이 $O(n^2)$으로 쓸 필요 없음
  - Tight하지 않은 만큼 정보의 손실이 발생 가능

차수 표기법: O(Big-Oh)-표기

- $5n^2 = O(n^2)$임을 보여라.
  - Proof: $c = 6, n_0 = 1$로 잡으면, $n \geq n_0$에 대하여 $5n^2 \leq 6n^2$이다. 즉 정의를 만족하는 c와 n_0가 존재한다.

- $5n^2 + 3 = O(n^2)$임을 보여라.
  - Proof: $c = 6, n_0 = 2$로 잡으면, $n \geq n_0$에 대하여 $5n^2 + 3 \leq 6n^2$이다. 즉 정의를 만족하는 c와 n_0가 존재한다.

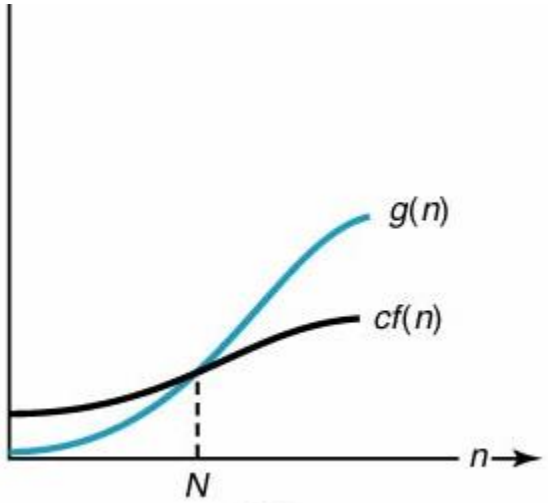
- $n^3 \in O(n^2)$?
  - $n \geq n_0$인 모든 n에 대해서 $n^3 \leq c \times n^2$이 성립하는 c와 n_0값은 존재하지 않는다. 즉, 양변을 n^2으로 나누면, $n \leq c$가 되는데 c를 아무리 크게 잡더라도 그 보다 더 큰 n이 존재한다.

✓ $O(f(n))$은 최고차 항의 차수가 $f(n)$과 일치하거나 더 작은 함수들의 집합

차수 표기법: Ω(Big-Omega)-표기

- Ω(Big-Omega)-표기
  - 정의: $n \geq n_0$인 모든 n에 대해서 $g(n) \geq c \times f(n)$을 만족하는 상수 c와 n_0가 존재하면, $g(n) \in \Omega(f(n))$이다.

  - $g(n)$이 $\Omega(n^2)$이고 $g(n)$이 어떤 알고리즘의 시간복잡도이면 그 알고리즘의 실행시간은 이차함수와 같거나 좋을 수 없다는 것을 의미한다.



(b) $g(n) \in \Omega(f(n))$

차수 표기법: Ω(Big-Omega)-표기

- $\Omega(f(n))$
  - 적어도 $f(n)$의 비율로 증가하는 함수들의 집합
  - 함수의 점근적 하한
  - $n^2,\ 3n^2-50,\ 5n^3+15,\ 2n^2\log n+1,\ n^2+\sqrt{n}=\Omega(n^2)$

- $5n^2=\Omega(n^2)$임을 보여라.
  - Proof: $c=4, n_0=1$로 잡으면, $n\geq n_0$에 대하여 $4n^2\leq 5n^2$이다. 즉 정의를 만족하는 c와 n_0가 존재한다.

- $5n^2+3=\Omega(n^2)$임을 보여라.
  - Proof: $c=1, n_0=1$로 잡으면, $n\geq n_0$에 대하여 $n^2\leq 5n^2+3$이다. 즉 정의를 만족하는 c와 n_0가 존재한다.

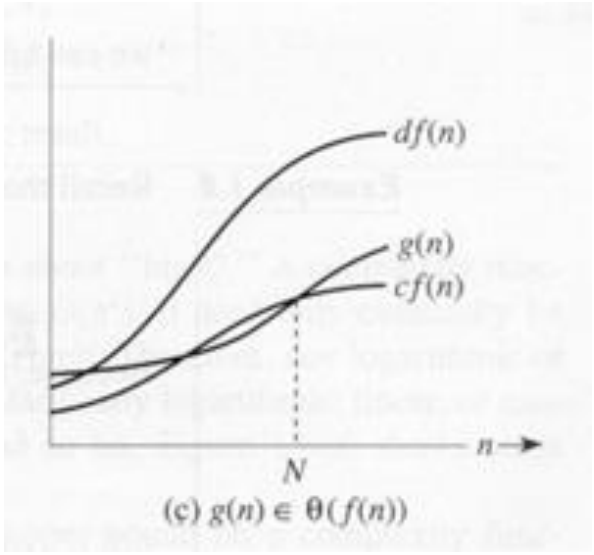
✓ $\Omega(f(n))$은 최고차 항의 차수가 $f(n)$과 같거나 그보다 큰 함수들의 집합

차수 표기법: Ω(Big-Omega)-표기

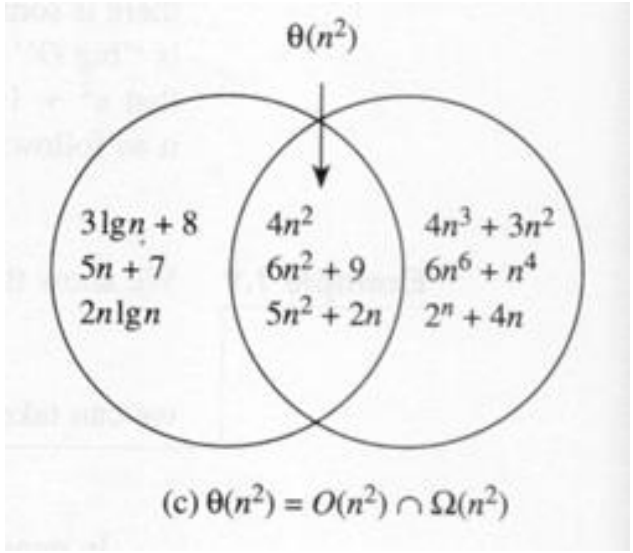
- $n \in \Omega(n^2)$?
  - Proof by contradiction: $n \in \Omega(n^2)$ 이라고 가정하면 $n \geq n_0$인 모든 n에 대해서, $n \geq c \times n^2$이 성립하는 실수 c와 n_0가 존재한다. 위의 부등식의 양변을 cn으로 나누면 $\frac{1}{c} \geq n$가 된다. 그러나 이 부등식은 절대로 성립할 수 없다. 따라서 위의 가정은 모순이다.

차수 표기법: Θ(Theta)-표기

- $\Theta(f(n))$
  - 정의: $n \geq n_0$인 모든 n에 대해서 $c \times f(n) \leq g(n) \leq d \times f(n)$이 성립하는 실수 c와 d, 그리고n_0가 존재하면 $g(n) \in \Theta(f(n))$이다.
  - 점근적 증가율이 $f(n)$과 일치하는 모든 함수들의 집합
  - $n^2, 3n^2 - 50, 7n^2 + 15, 2n^2 + 3n\log n, n^2 + \sqrt{n} = \Theta(n^2)$
  - $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$



(c) $g(n) \in \Theta(f(n))$



(c) $\Theta(n^2) = O(n^2) \cap \Omega(n^2)$

차수 표기법: Θ(Theta)-표기

- $5n^2 = \Theta(n^2)$임을 보여라.
  - Proof: 앞의 예제들에서 $5n^2 = O(n^2)$와 $5n^2 = \Omega(n^2)$임을 각각 증명하였다. 그러므로 $5n^2 = \Theta(n^2)$이다.

- $5n^2 + 3 = \Theta(n^2)$임을 보여라.
  - Proof: 앞의 예제들에서 $5n^2 + 3 = O(n^2)$와 $5n^2 + 3 = \Omega(n^2)$임을 각각 증명하였다. 그러므로 $5n^2 + 3 = \Theta(n^2)$이다.

✓ $\Theta(f(n))$은 최고차 항의 차수가 $f(n)$과 일치하는 함수들의 집합

로피탈의 법칙을 이용한 효율성 비교

- 로피탈의 법칙 (L'Hôspital's Rule)
  - $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ 이면 $\lim_{n\to\infty} \frac{g(n)}{f(n)} = \lim_{n\to\infty} \left(\frac{g'(n)}{f'(n)}\right)$
- 로피탈의 법칙을 이용한 알고리즘의 효율성 비교
  - 최악의 경우 시간 복잡도가 각각 $W_C(n) = \Theta(\sqrt{n})$, $W_D(n) = \Theta(\log_2 n)$ 인 알고리즘 C와 D의 효율성 비교
    - n 값에 따른 $\sqrt{n}$과 $\log_2 n$ 값의 변화

n	16	64	256	1,024	4,096	
$\sqrt{n}$	4	8	16	32	64	…
$\log_2 n$	4	6	8	10	12	…

    - $\lim_{n\to\infty} W_C(n) = \lim_{n\to\infty} W_D(n) = \infty$
    - $W'_C(n) = (\sqrt{n})' = \frac{1}{2} n^{\frac{1}{2}-1} = \frac{1}{2\sqrt{n}}$
    - $W'_D(n) = (\log_2 n)' = \frac{1}{n \ln 2}$
    - $\lim_{n\to\infty} \frac{W_C(n)}{W_D(n)} = \lim_{n\to\infty} \frac{\sqrt{n}}{\log_2 n} = \lim_{n\to\infty} \frac{n \ln 2}{2\sqrt{n}} = \lim_{n\to\infty} \frac{\sqrt{n} \ln 2}{2} = \infty$
    - 따라서 $\log_2 n$이 $\sqrt{n}$보다 더 효율적이다.

Q & A

Thank you!