ALGORITHM
알고리즘

Lecture **02**

알고리즘 소개 (2)

컴퓨터과학과 | **이관용 교수**

한국방송통신대학교

학습목차

01 알고리즘 분석

알고리즘 분석

정확성 분석

- 유효한 입력에 대해 유한 시간 내에 정확한 결과의 생성 여부
  - ✓ 수학적 기법을 사용한 이론적인 증명 과정

효율성 분석

- 알고리즘 수행에 필요한 컴퓨터 자원의 양을 측정/평가
- 공간 복잡도 space complexity
  - ✓ 메모리의 양 = 정적 공간 + 동적 공간
- 시간 복잡도 time complexity
  - ✓ 수행시간 = 알고리즘의 실행에서부터 완료까지 걸리는 시간

시간 복잡도

컴퓨터에서 실행시켜 실제 수행시간을 측정하는 방법?

- 실행 환경에 종속적이므로 일반성이 결여된 방법
  - ✓ 컴퓨터 속도, 구현에 사용된 프로그래밍 언어, 프로그램 작성 방법, 컴파일러의 효율성 등에 따라 시간이 달라짐

알고리즘 수행시간 = Σ{각 문장(연산)이 수행되는 횟수}

- 수행시간에 영향을 미치는 요인
  - ✓ 입력 크기
    - → "입력되는 데이터의 크기", "문제가 해결하려는 대상이 되는 개체의 개수"
      - → 예: 리스트 원소의 개수, 행렬의 크기, 그래프의 정점의 수 등
  - ✓ 입력 데이터의 상태

시간 복잡도

입력 크기 n이 커질수록 수행시간도 증가

- 단순히 단위 연산이 수행되는 개수의 합으로 표현하는 것은 부적절

→ 입력 크기 n의 함수 $f(n)$으로 표현

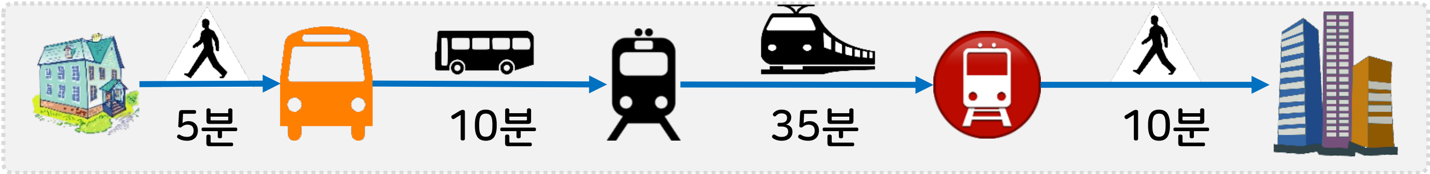
입력 데이터의 상태에 종속적

S_n : 크기 n인 입력들의 집합
$P(I)$: 입력 I가 발생할 확률
$t(I)$: 입력 I일 때 알고리즘의 수행시간

- 평균 수행시간 → $A(n) = \sum_{I \in S_n} P(I)t(I)$
- 최선 수행시간 → $B(n) = \min_{I \in S_n} t(I)$
- 최악 수행시간 → $W(n) = \max_{I \in S_n} t(I)$

시간 복잡도

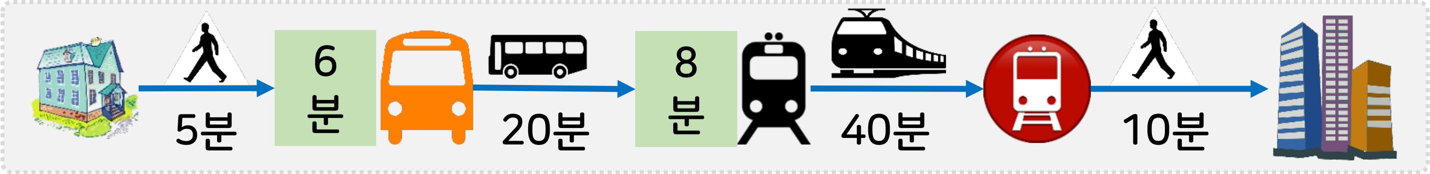
최선의 경우



60분

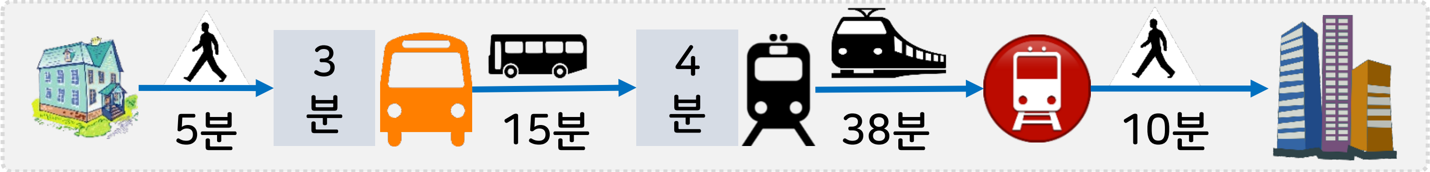
최악의 경우

→ **"늦어도 89분이면 출근한다."**



89분

평균의 경우



75분

시간 복잡도


```
SumAverage( A[], n )
{ //A[0..n-1], n : 입력 배열과 데이터 개수
  sum = 0;
  i = 0;
  while ( i < n ) {
     sum = sum + A[i];
     i = i + 1;
  }
  average = sum / n;
  print sum, average;
}
```

입력 크기

sum = 0;	1
i = 0;	1
while (i < n) {	n+1
sum = sum + A[i];	n
i = i + 1;	n
average = sum / n;	1
print sum, average;	1

$T(n) = 3n + 5 \xrightarrow{\text{점근성능}} O(n)$

Big-oh 표기

02 점근성능

점근성능

입력 크기 n이 무한히 커짐에 따라 결정되는 성능

	$f_1(n)=10n+9$	$f_2(n)=n^2/2+3n$
n=5	59	27.5
n=10	109	80
n=15	159	157.5
n=16	169	176
n=20	209	260
⋮	⋮	⋮

$f_3(n)=2n^2+5n+200$

n=1	2+5+200 = **207**
n=5	50+25+200 = **275**
n=10	200+50+200 = **450**
n=20	800+100+200 = **1,100**
	⋮
n=1000	2,000,000+5,000+200 = **2,005,200**
	⋮

점근성능

점근성능의 결정 방법

- 입력 크기가 충분히 커짐에 따라 함숫값에 가장 큰 영향을 미치는 차수를 찾음
- 수행시간의 다항식 함수에서 최고차항만을 계수 없이 취해서 표현
  - ✓ 수행시간의 정확한 값이 아닌 어림값
    - → 수행시간의 증가 추세를 파악하는 것이 쉬움
    - → 알고리즘 간의 우열을 따질 때 유용

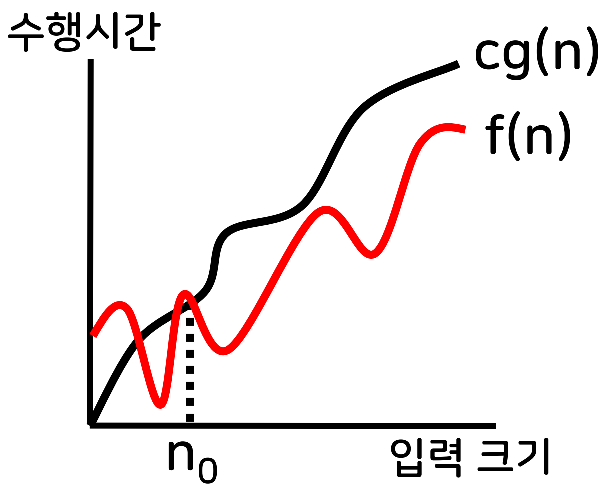
점근성능의 표기법

정의 1 'Big-oh' 점근적 상한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

어떤 양의 상수 c와 n_0이 존재하여

모든 $n \geq n_0$에 대하여 $f(n) \leq c \cdot g(n)$이면 $f(n) = O(g(n))$이다.



$$f(n) = O(g(n))$$

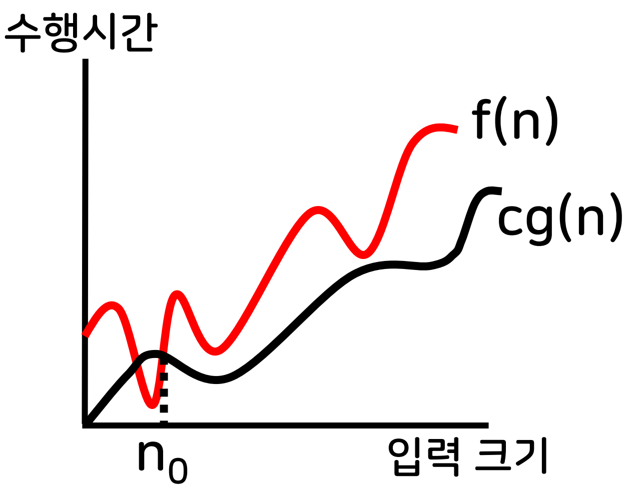
점근성능의 표기법

정의 2 'Big-omega' 점근적 하한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

어떤 양의 상수 c와 n_0이 존재하여

모든 $n \geq n_0$에 대하여 $f(n) \geq c \cdot g(n)$이면 $f(n) = \Omega(g(n))$이다.



$$f(n) = \Omega(g(n))$$

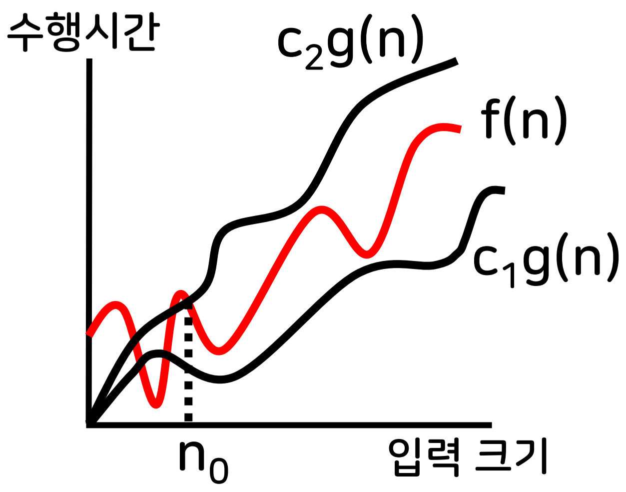
점근성능의 표기법

정의 3 'Big-theta' 점근적 상하한

함수 f와 g를 각각 양의 정수를 갖는 함수라 하자.

어떤 양의 상수 c_1, c_2와 n_0이 존재하여

모든 $n \geq n_0$에 대하여 $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$이면 $f(n)=\Theta(g(n))$이다.



$$f(n) = \Theta(g(n))$$

점근성능의 표기법

$f(n)=3n+3$, $g(n)=n$

- $c=5$, $n_0=2$이 존재하여 $n\geq2$인 모든 n에 대해서
  $f(n)\leq c\cdot g(n)$, 즉 $3n+3\leq5\cdot n$을 만족 → $f(n)=O(g(n))=O(n)$
- $c=3$, $n_0=1$이면 $n\geq1$에 대해서 $3n+3\geq3\cdot n$을 만족 → $f(n)=\Omega(g(n))=\Omega(n)$
- $f(n)=O(n)$이면서 $f(n)=\Omega(n)$ → $f(n)=\Theta(n)$

$f(n)=3n^3+3n-1$, $g(n)=n^3$

- $c=7$, $n_0=1$이면 $n\geq1$에 대해서 $f(n)\leq7\cdot g(n)$을 만족 → $f(n)=O(g(n))=O(n^3)$
- $c=2$, $n_0=2$이면 $n\geq2$에 대해서 $f(n)\geq2\cdot g(n)$을 만족 → $f(n)=\Omega(g(n))=\Omega(n^3)$
- $f(n)=O(n^3)$이면서 $f(n)=\Omega(n^3)$ → $f(n)=\Theta(n^3)$

점근성능의 표기법

- $f(n)=3n^3+3n-1 = O(g(n)) = O(n^3)$

	가장 낮은 차수의 함수
n^3 →	$O(n^3)$
n^4 →	$O(n^4)$
n^5 →	$O(n^5)$
… →	…

	가장 높은 차수의 함수
n^3 →	$\Omega(n^3)$
n^2 →	$\Omega(n^2)$
$n\log n$ →	$\Omega(n\log n)$
n →	$\Omega(n)$
$\log n$ →	$\Omega(\log n)$

- $f(n)=3n^3+3n-1 = \Omega(g(n)) = \Omega(n^3)$

점근성능의 표기법

주요 O-표기 간 연산 시간의 크기 관계

$f_1(n)=10n+9$ → $O(n)$

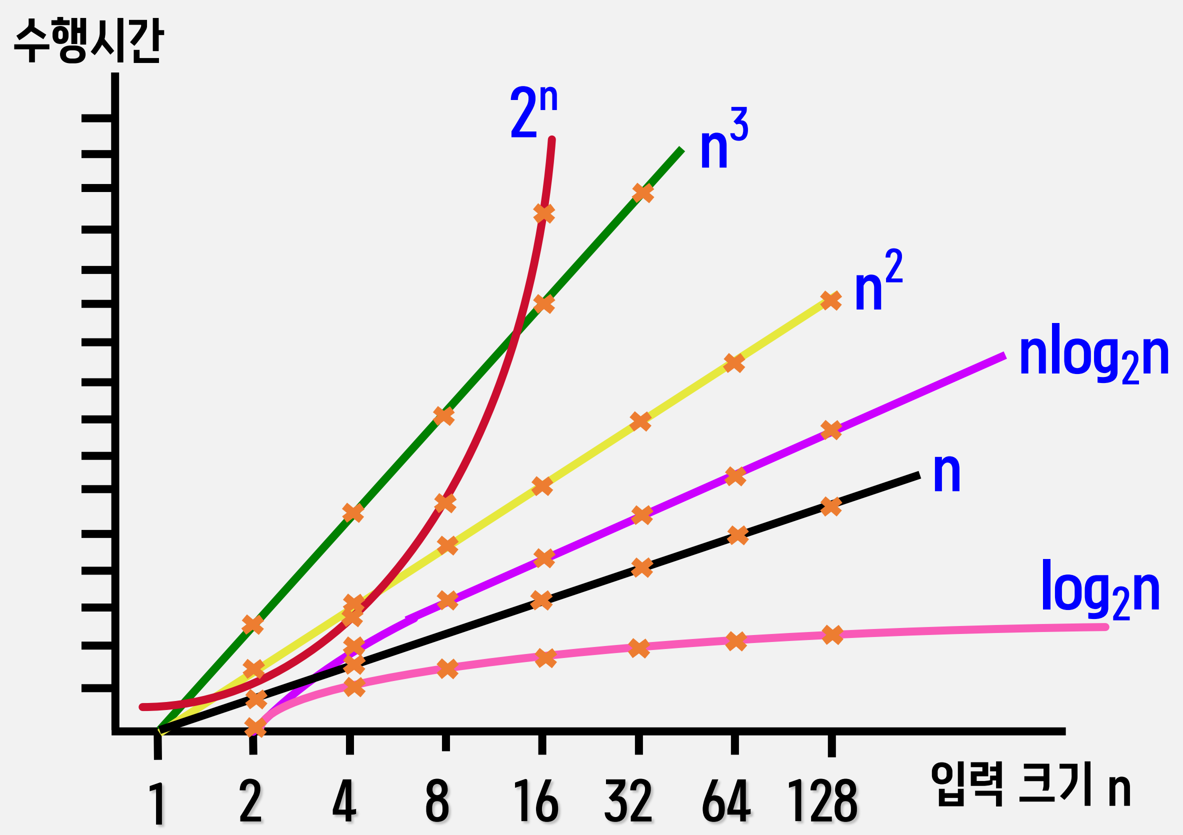
$f_2(n)= n^2/2+3n$ → $O(n^2)$

$f_3(n)= 3n^3+3n+2$ → $O(n^3)$

상수 시간		로그 시간		선형 시간		로그 선형 시간		제곱 시간		세제곱 시간		지수 시간
$O(1)$	<	$O(\log n)$	<	$O(n)$	<	$O(n\log n)$	<	$O(n^2)$	<	$O(n^3)$	<	$O(2^n)$

← 효율적 　　　　 비효율적 →

빅오 함수에 따른 연산 시간의 증가 추세



빅오 함수에 따른 연산 시간의 증가 추세

$\log n$	n	$n\log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4,096	65,536
5	32	160	1,024	32,768	4,294,967,296

알고리즘의 시간 복잡도 구하기

알고리즘의 시간 복잡도를 구하려면

- 기본 연산의 수행 횟수의 합 f(n)을 구한 후,
- f(n)=O(g(n))을 만족하는 최소 차수의 함수 g(n)을 찾음

실용적인 접근 방법

- 알고리즘의 모든 문장이 아닌 **루프의 반복 횟수**만을 조사하여 최고 차수를 시간 복잡도로 취함

→ O(최고차수)

알고리즘의 시간 복잡도 구하기


```
i = 1;              1
while (i <= n) {    n+1
  x = x + 1;        n
  i = i + 1;        n
}
```

$f(n)=3n+2$

$O(n)$

```
count = 0;                 1
for (i=0; i<n; i++)        n+1
    for (j=0; j<n; j++)    n+1
        count ++;          n^2
print count;               1
```

$f(n)=n^2+2n+4$

$O(n^2)$

03 순환 알고리즘

순환 알고리즘의 성능

순환 recursion, 재귀 알고리즘

- 알고리즘의 수행 과정에서 자기 자신의 알고리즘을 다시 수행하는 형태

```
BinarySearch (A[ ], key, Left, Right)  {
    if (Left > Right) return (-1);
    mid = ⌊(Left + Right)/2⌋;
    if (A[Mid] == key) return (Mid);
    else if (key < A[Mid]) BinarySearch(A, key, Left, Mid-1)
            else BinarySearch(A, key, Mid+1, Right);
}
```

$T(n) = T(n/2) + O(1), \ T(1)=c_1$

점화식의 폐쇄형 구하기

$$T(n) = T\left(\frac{n}{2}\right) + c_2, \qquad T(1) = c_1$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + c_2 \\
&= T\left(\frac{n}{4}\right) + c_2 + c_2 = T\left(\frac{n}{2^2}\right) + 2c_2 \\
&= T\left(\frac{n}{8}\right) + c_2 + c_2 + c_2 = T\left(\frac{n}{2^3}\right) + 3c_2 \\
&\quad \cdots \\
&= T\left(\frac{n}{2^{k-1}}\right) + (k-1)c_2 \\
&= T\left(\frac{n}{2^k}\right) + kc_2 \\
&= T(1) + kc_2 \quad (n = 2^k \text{인 경우에만 정의, } k = \log_2 n) \\
&= c_1 + c_2 \log_2 n \\
&= \Theta(\log_2 n)
\end{aligned}$$

기본 점화식과 폐쇄형

(1) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n \geq 2 \end{cases}$ → $T(n) = \Theta(n)$

(2) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(n), & n \geq 2 \end{cases}$ → $T(n) = \Theta(n^2)$ 퀵 정렬 (최악의 경우)

(3) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ T\left(\frac{n}{2}\right) + \Theta(1), & n \geq 2 \end{cases}$ → $T(n) = \Theta(\log n)$ 이진 탐색

(4) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ T\left(\frac{n}{2}\right) + \Theta(n), & n \geq 2 \end{cases}$ → $T(n) = \Theta(n)$

(5) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(1), & n \geq 2 \end{cases}$ → $T(n) = \Theta(n)$

(6) $T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n \geq 2 \end{cases}$ → $T(n) = \Theta(n\log n)$ 합병 정렬
퀵 정렬 (최선의 경우)

효율적인 알고리즘의 중요성

n	f(n)=logn	f(n)=n	f(n)=nlogn	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 분
50	0.006 μs	0.05 μs	0.282 μs	2.50 μs	125.0 μs	13일
10^2	0.007 μs	0.10 μs	0.664 μs	10.00 μs	1.0 ms	4×10^{13} 년
10^3					1.0 s	
10^4					16.7 분	
10^5					11.6 일	
10^6	0.020 μs	1.00 ms	19.930 ms	16.70 분	31.7 년	
10^7	0.023 μs	0.01 s	2.660 s	1.16 일	31,709 년	
10^8	0.027 μs	0.10 s	2.660 s	115.70 일	3.17×10^7 년	
10^9	0.030 μs	1.00 s	29.900 s	31.70 년		

입력 크기가 커질수록 수행시간의 차이가 확연

→ 효율적인 알고리즘의 설계/선택이 중요

단위 연산의
 수행 시간 = 1ns

1ns = 10^{-9}초

1μs = 10^{-6}초

1ms = 10^{-3}초

정리하기

1. 알고리즘 분석

- 정확성 분석 + 효율성 분석
- 효율성 분석 → 공간 복잡도, 시간 복잡도(각 문장의 수행횟수의 합)
- 시간 복잡도 → 입력 크기의 함수와 최악의 수행시간으로 표현

2. 점근성능

- 입력 크기가 무한히 커짐에 따라 결정되는 성능 → 최고차수만으로 간략히 표현
- 표기법 → O-표기, Ω-표기, θ-표기
- $O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$

3. 순환 알고리즘

- 순환 알고리즘의 수행시간은 점화식 형태로 정의되며, 폐쇄형을 구해서 표현
- 분할정복 방법이 적용된 알고리즘은 기본적으로 순환 알고리즘 형태를 가짐
- 기본 점화식과 폐쇄형

ALGORITHM 알고리즘

다음시간에는

Lecture 03

정렬 (1)

컴퓨터과학과 | 이관용 교수