1장. 알고리즘 설계와 분석의 기초

http://academy.hanb.co.kr

1장. 알고리즘 설계와 분석의 기초

생각하는 방법을 터득한 것은
미래의 문제를 미리 해결한 것이다.

– 제임스 왓슨

학습목표

- 알고리즘의 정의와 필요성을 인지한다.
- 아주 기초적인 알고리즘 수행시간 분석을 할 수 있도록 한다.
- 점근적 표기법을 이해한다.

알고리즘이란 무엇인가?

- 문제 해결 절차를 체계적으로 기술한 것
- 문제의 요구조건
  - 입력과 출력으로 명시할 수 있다
  - 알고리즘은 입력으로부터 출력을 만드는 과정을 기술

입출력의 예

- 문제
  - 100명의 학생의 시험점수의 최대값을 찾으라
- 입력
  - 100명의 학생들의 시험점수
- 출력
  - 위 100개의 시험점수들 중 최대값

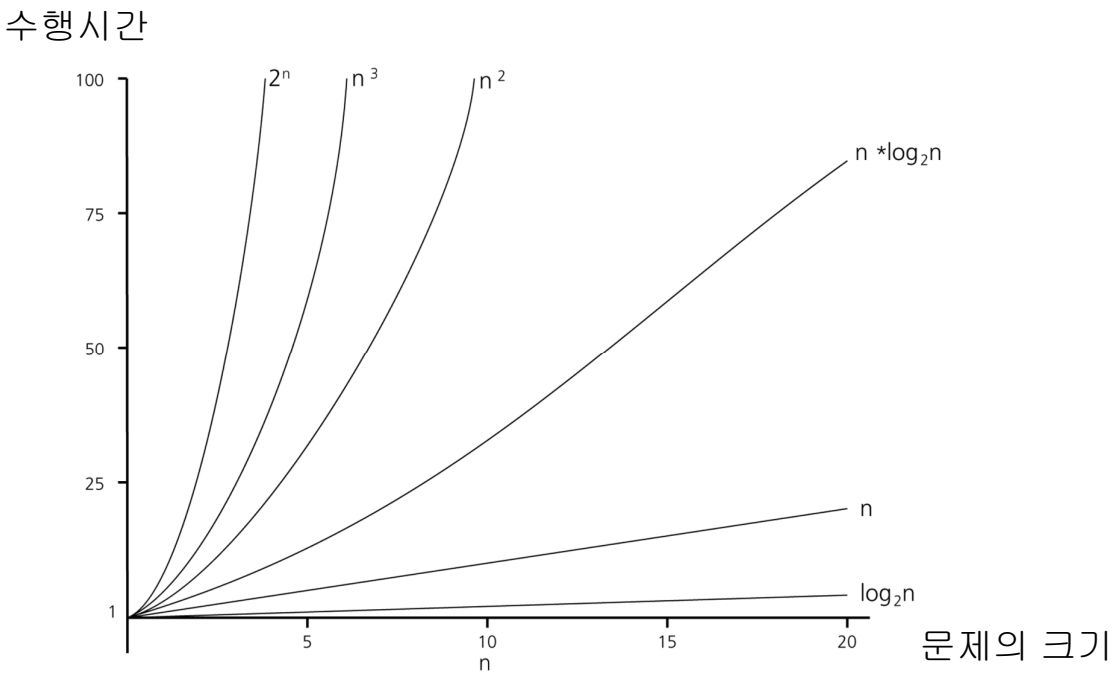
알고리즘 공부의 목적

- 특정한 문제를 위한 알고리즘의 습득
- 체계적으로 생각하는 훈련
- 지적 추상화의 레벨 향상
  - Intellectual abstraction
  - 연구나 개발에 있어 정신적 여유를 유지하기 위해 매우 중요한 요소

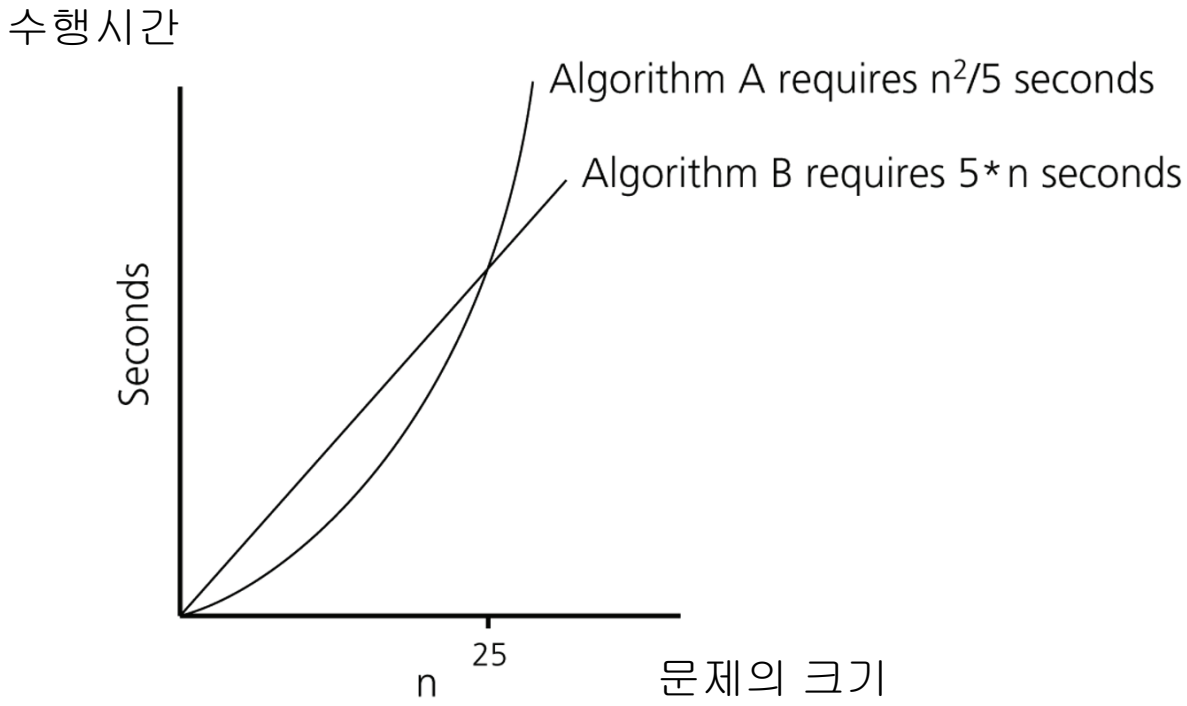
바람직한 알고리즘

- 명확해야 한다
  - 이해하기 쉽고 가능하면 간명하도록
  - 지나친 기호적 표현은 오히려 명확성을 떨어뜨림
  - 명확성을 해치지 않으면 일반언어의 사용도 무방
- 효율적이어야 한다
  - 같은 문제를 해결하는 알고리즘들의 수행시간이 수백만배 이상 차이가 날 수 있다

알고리즘의 수행시간



알고리즘의 수행시간



알고리즘의 수행시간

	n					
Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

알고리즘의 수행시간

- 알고리즘의 수행시간을 좌우하는 기준은 다양하게 잡을 수 있다
  - 예: for 루프의 반복횟수, 특정한 행이 수행되는 횟수, 함수의 호출횟수, …
- 몇 가지 간단한 경우의 예를 통해 알고리즘의 수행시간을 살펴본다

알고리즘의 수행시간

```
sample1(A[ ], n)
{
        k = n/2 ;
        return A[k] ;
}
```

✓ n에 관계없이 상수 시간이 소요된다.

알고리즘의 수행시간

```
sample2(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                sum← sum+ A[i] ;
        return sum ;
}
```

✓ n에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample3(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                for j ← 1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

✓ n^2에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample4(A[ ], n)
{
    sum ← 0 ;
    for i ← 1 to n
        for j ← 1 to n {
            k ← A[1 ... n] 에서 임의로 n/2 개를 뽑을 때 이 중 최대값 ;
            sum ← sum + k ;
        }
    return sum ;
}
```

✓ n^3에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
sample5(A[ ], n)
{
    sum ← 0 ;
    for  i ← 1 to n-1
        for  j ←  i+1 to n
            sum← sum+ A[i]*A[j] ;
    return sum ;
}
```

✓ $(n\text{-}1) + (n\text{-}2) + \ldots + 2 + 1 = \frac{n(n-1)}{2}$

➔ n^2에 비례하는 시간이 소요된다.

알고리즘의 수행시간

```
factorial(n)
{
        if (n==1) return 1 ;
        return n*factorial(n-1) ;
}
```

✓ factorial()이 몇 번 호출되는가?
➔ n에 비례하는 시간이 소요된다.

재귀와 귀납적 사고

- **재귀=자기호출(recursion)**
- **재귀적 구조**
  - 어떤 문제 안에 크기만 다를 뿐 성격이 똑같은 작은 문제(들)가 포함되어 있는 것 ➔ 수학적 귀납법과 관련!
    - 자신보다 작은 문제에 대해서는 이 알고리즘이 제대로 작동한다고 가정하는 것!
  - 예1: factorial
    - $N! = N \times (N-1)!$
  - 예1: 수열의 점화식
    - $a_n = a_{n-1} + 2$

재귀의 예: Mergesort (병합정렬)

```
mergeSort(A[ ], p, r)
▷ A[p ... r]을 정렬한다
{
    if (p < r) then {
        q ← ⌊(p+r)/2⌋;  ---------- ①  ▷ p, q의 중간 지점 계산
        mergeSort(A, p, q);  ---------- ②  ▷ 전반부 정렬
        mergeSort(A, q+1, r); ---------- ③  ▷ 후반부 정렬
        merge(A, p, q, r);  ---------- ④  ▷ 병합
    }
}

merge(A[ ], p, q, r)
{
    정렬되어 있는 두 배열 A[p ... q]와 A[q+1 ... r]을 합하여
    정렬된 하나의 배열 A[p ... r]을 만든다.
}
```



```
mergeSort(A[ ], p, r)
▷ A[p ... r]을 정렬한다
{
    if (p < r) then {
        q ← (p+r)/2;  ---------- ①  ▷ p, q의 중간 지점 계산
        mergeSort(A, p, q);  ---------- ②  ▷ 전반부 정렬
        mergeSort(A, q+1, r); ---------- ③  ▷ 후반부 정렬
        merge(A, p, q, r);  ---------- ④  ▷ 병합
    }
}
```

✓ ②, ③은 자기호출
✓ ①, ④는 크기가 n인 배열과 이를 이등분한 배열간의 관계를 반영

다양한 알고리즘의 적용 주제들

- 카 네비게이션
- 스케줄링
  - TSP(Traveling Salesman Problem), 차량 라우팅, 작업공정, …
- Human Genome Project
  - DNA간 관계 파악, 유사성 매칭, 진화 계통도, …
- 검색
  - 데이터베이스, 웹 페이지들, …
- 자원의 배치
- 반도체 설계
  - Partitioning, placement, routing, …
- …

알고리즘을 왜 분석하는가?

- **무결성 확인**
- **자원 사용의 효율성 파악**
  - 자원
    - 시간
    - 메모리, 통신대역, …

알고리즘의 분석

- 크기가 작은 문제
  - 알고리즘의 효율성이 중요하지 않다
  - 비효율적인 알고리즘도 무방
- 크기가 충분히 큰 문제
  - 알고리즘의 효율성이 중요하다
  - 비효율적인 알고리즘은 치명적
- 입력의 크기가 충분히 큰 경우에 대한 분석을 점근적 분석이라 한다

점근적 분석 Asymptotic Analysis

- 입력의 크기가 충분히 큰 경우에 대한 분석
- 이미 알고 있는 점근적 개념의 예

$$\lim_{n \to \infty} f(n)$$

- O, Ω, Θ, ω, o 표기법

점근법 표기법 Asymptotic Notations

- $O(f(n))$
  - 기껏해야 $f(n)$의 비율로 증가하는 함수
  - e.g., $O(n)$, $O(n\log n)$, $O(n^2)$, $O(2^n)$, …
- Formal definition
  - $O(f(n)) = \{ g(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t.} \forall n \geq n_0, cf(n) \geq g(n) \}$
  - $g(n) \in O(f(n))$을 관행적으로 $g(n) = O(f(n))$이라고 쓴다.
- 직관적 의미
  - $g(n) = O(f(n)) \Rightarrow g$ 는 f 보다 빠르게 증가하지 않는다
  - 상수 비율의 차이는 무시(교재 31p. 표 1-1 참조)
  - 점근적 상한을 의미

점근적 표기법

- 예, $O(n^2)$
  - $3n^2 + 2n$
  - $7n^2 - 100n$
  - $n\log n + 5n$
  - $3n$
- 알 수 있는 한 최대한 tight 하게
  - $n\log n + 5n = O(n\log n)$ 인데 굳이 $O(n^2)$으로 쓸 필요 없다.
  - Tight하지 않은 만큼 정보의 손실이 일어나므로.

점근적 표기법

- $\Omega(f(n))$
  - 적어도 $f(n)$의 비율로 증가하는 함수
  - $O(f(n))$과 대칭적
- Formal definition
  - $\Omega(f(n)) = \{ g(n) \mid \exists c > 0, n_0 \geq 0 \text{ s.t. } \forall n \geq n_0, cf(n) \leq g(n) \}$
- 직관적 의미
  - $g(n) = \Omega(f(n)) \Rightarrow g$ 는 f 보다 느리게 증가하지 않는다.
  - 점근적 하한
- 예: $\Omega(n^2)$
  - $n^2, 3n^2-50, 5n^3+15$
  - $2n^2\log n+1, n^2+\sqrt{n}$

점근적 표기법

- $\Theta(f(n))$
  - 점근적 증가율이 $f(n)$과 일치하는 함수
- Formal definition
  - $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- 직관적 의미
  - $g(n) = \Theta(f(n)) \Rightarrow g$ 는 f 와 같은 정도로 증가한다.
  - $\Theta(f(n))$는 최고차항의 차수가 $f(n)$과 일치하는 함수들의 집합

각 점근적 표기법의 직관적 의미

- $O(f(n))$
  - Tight *or* loose upper bound
- $\Omega(f(n))$
  - Tight *or* loose lower bound
- $\Theta(f(n))$
  - Tight bound

점근적 복잡도의 예

- 정렬 알고리즘들의 복잡도 표현 예 (3장에서 공부함)
  - 선택정렬
    - $\Theta(n^2)$
  - 힙정렬
    - $O(n\log n)$
  - 퀵정렬
    - $O(n^2)$
    - 평균 $\Theta(n\log n)$

시간 복잡도 분석의 종류

- Worst-case
  - Analysis for the worst-case input(s)
- Average-case
  - Analysis for all inputs
  - More difficult to analyze
- Best-case
  - Analysis for the best-case input(s)
  - 별 유용하지 않음

저장/검색의 복잡도

- 배열
  - $O(n)$
- Binary search trees
  - 최악의 경우 $\Theta(n)$
  - 평균 $\Theta(\log n)$
- Balanced binary search trees
  - 최악의 경우 $\Theta(\log n)$
- B-trees
  - 최악의 경우 $\Theta(\log n)$
- Hash table
  - 평균 $\Theta(1)$

크기 n인 배열에서 원소 찾기

- Sequential search
  - 배열이 아무렇게나 저장되어 있을 때
  - Worst case: $\Theta(n)$
  - Average case: $\Theta(n)$
- Binary search
  - 배열이 정렬되어 있을 때
  - Worst case: $\Theta(\log n)$
  - Average case: $\Theta(\log n)$

Thank you