자료구조 Data Structure | 조행래

자료구조의 기본 개념

알고리즘의 복잡도 계산

학습 목표

- 알고리즘의 복잡도를 정의할 수 있다.
- 알고리즘의 복잡도를 표현할 수 있다.
- 알고리즘의 복잡도를 계산할 수 있다.

1. 성능 분석

- **프로그램의 평가 기준**
  - 주어진 문제를 해결
  - 정확성(Correctness)
  - 문서화(Documentation)
  - 모듈화(Modularization)
  - 가독성(Readability)
  - 공간 효율(Space efficiency)
  - 시간 효율(Time efficiency)

  주어진 문제를 해결, 정확성(Correctness) } 필수적인 요소

  문서화(Documentation), 모듈화(Modularization), 가독성(Readability) } 좋은 프로그래밍 습관

  공간 효율(Space efficiency), 시간 효율(Time efficiency) } 성능과 관련

- **성능 분석(Complexity theory, Simulation) vs. 성능 측정(Benchmarking)**

- **복잡도(Complexity)의 정의**
  - 공간 복잡도: 프로그램 실행에 사용되는 메모리
  - 시간 복잡도: 프로그램 실행에 걸리는 시간

2. 시간 복잡도(Time Complexity)

- **실행에 걸리는 시간(T_p) = 컴파일 시간 + 실행 시간**
  - 컴파일 시간은 고정 & 한번만 필요
  - 질문: T_p 를 어떻게 계산할까?
    - T_p는 컴파일러 option과 하드웨어 사양에 따라 가변
    - 프로그램 단계 수를 활용하자.

- **프로그램 단계 수(Program Step)**
  - 정의: 실행 시간이 프로그램의 특성과는 무관한 프로그램의 문법적인 혹은 논리적인 단위
  - 프로그램 단계 수의 계산
    - 프로그램에 count를 증가시키는 문장을 추가
    - 테이블 방식을 이용

테이블 방식의 예(1)

Statement	s/e	Frequency	Total step
Float sum(float list[ ], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for (i = 0; i < n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			**2n+3**

* s/e (steps/execution): 명령문에 대한 단계 수

테이블 방식의 예(2)

Statement	s/e	Frequency	Total step
Float rsum(float list[ ], int n)	0	0	0
{	0	0	0
if (n)	1	n+1	n+1
return rsum(list, n-1)+ list[n-1];	1	n	n
return list[0];	1	1	1
}	0	0	0
Total			**2n+2**

테이블 방식의 예(3)

Statement	s/e	Frequency	Total step
void add(int a [ ] [MAX_SIZE] …)	0	0	0
{	0	0	0
int i, j;	0	0	0
for (i = 0; i < rows; i++)	1	rows+1	rows+1
for (j = 0; j < cols; j++)	1	rows · (cols+1)	rows · cols + rows
c[i][j] = a[i][j] + b[i][j];	1	rows · cols	rows · cols
}	0	0	0
Total			**2rows · cols + 2rows + 1**

3. 점근 표기법(O, Ω, Θ)

- **동기**
  - 정확한 프로그램 단계 수를 계산하는 것은 쉽지 않다.
  - 프로그램 단계 수의 정의 자체가 정확하지 않다.
  - 100n + 10과 30n + 30의 비교

- **접근 방법**
  - $T_p(n) = n^2 + 100n$ 이라고 가정
  - n이 충분히 클 경우, 임의의 c_3에 대해 $T_p(n) > c_3n$

Big-Oh (O)

- **정의(Big-Oh): f(n) = O(g(n)) iff**
  - ∃(c and n_0 > 0) such that
  - f(n) ≤ **c**g(n) for all n, n ≥ **n_0**.

- **Example**
  - 3n + 2 = O(n) as 3n + 2 ≤ 4n for all n ≥ 2
  - 100n + 6 = O(n) as 100n + 6 ≤ 101n for all n ≥ 6
  - $10n^2 + 4n = O(n^2)$ as $10n^2 + 4n \leq 11n^2$ for all n ≥ 5
  - $6*2^n + n^2 = O(2^n)$ as $6*2^n + n^2 \leq 7*2^n$ for all n ≥ 4
  - $3n + 2 = O(n^2)$, $10n^2 + 4n = O(n^3)$

Omega (Ω)

- **정의(Omega):** $f(n) = \Omega(g(n))$ iff
  - $\exists$(c and $n_0 > 0$) such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$.
  - g(n)은 f(n)의 lower bound

- **Example**
  - $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for all $n \geq 1$
  - $10n^2 + 4n = \Omega(n^2)$ as $10n^2 + 4n \geq 10n^2$, $\forall n \geq 1$
  - $6*2^n + n^2 = \Omega(2^n)$ as $6*2^n + n^2 \geq 6*2^n$, $\forall n \geq 1$
  - $3n + 2 = \Omega(1)$, $10n^2 + 4n = \Omega(n)$, $6*2^n + n^2 = \Omega(n^2)$

Theta (Θ)

- **정의(Theta):** $f(n) = \Theta(g(n))$ iff
  - $\exists$ (c_1, c_2, and $n_0 > 0$) such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.
  - g(n)은 f(n)의 lower bound이면서 upper bound임

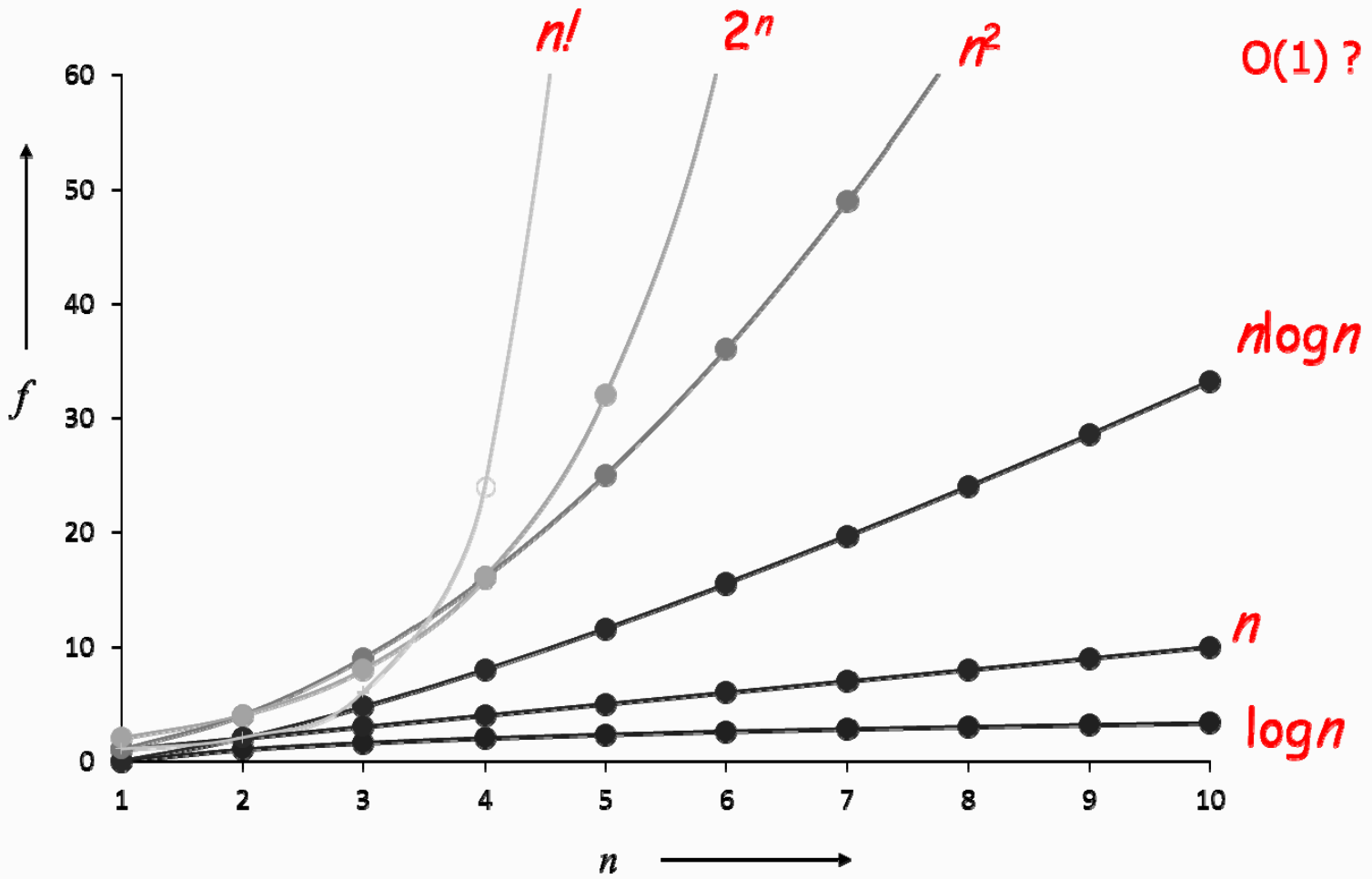
- **Example**
  - $3n + 2 = \Theta(n)$ as $3n + 2 \geq 3n$ for all $n \geq 2$ and $3n + 2 \leq 4n$ for all $n \geq 2$
  - $10n^2 + 4n + 2 = \Theta(n^2)$, $6*2^n + n^2 = \Theta(2^n)$
  - $3n + 2 \neq \Theta(1)$, $3n + 2 \neq \Theta(n^2)$

예제 알고리즘들의 근사 표현

- $T_{sum}(n) = 2n + 3 = \Theta(n)$
- $T_{rsum}(n) = 2n + 2 = \Theta(n)$
- $T_{add}(row, col) = 2row * col + 2row + 1 = \Theta(row * col)$

- **이진 검색**: $\Theta(\log_2 n)$
  - $T(n) = T(n/2) + 1$
  - $T(1) = 1$

대표적인 복잡도 함수의 그래프



요약 정리

- 복잡도의 정의를 이해
- 알고리즘의 시간 복잡도를 분석하는 방법 이해
- 복잡도를 점근 표기법으로 표현하는 방법을 이해