COSE213: Data Structure

Lecture 2 - 알고리즘 분석 (Analysis of Algorithms)

Minseok Jeon

2024 Fall

알고리즘

- 알고리즘: 특정 문제를 해결하기 위해 명확하게 정의된 일련의 계산 절차 및 규칙
- 주요 특징
  - 입력(input): 알고리즘은 0개 이상의 입력을 받음
  - 출력(output): 알고리즘은 적어도 하나의 출력을 생성함
  - 명확성(Definiteness) : 각 단계는 명확하게 정의되어야 함
  - 유한성(Finiteness) : 알고리즘은 유한한 수의 단계 후에 종료되어야 함
  - 효율성 (Effectiveness) : 각 명령은 충분히 기본적이어서 사람이 쉽게 따라갈 수 있어야 함

https://www.merriam-webster.com/dictionary/algorithm
https://www.geeksforgeeks.org/introduction-to-algorithms/

알고리즘 표현방법

- 알고리즘을 표현하는 방법은 여러 가지가 있으며, 각 방법에는 장단점이 있음
    - 자연어 (Natural Language)
    - 프로그래밍 언어 (Programming Language)
    - 순서도 (Flowchart)
    - 슈도코드 (Pseudocode)

알고리즘 표현 방법 1: 자연어 (Natural Language)

- 알고리즘의 단계별 프로세스를 직관적인 자연어로 설명함

(1) 학생들의 점수의 배열을 입력으로 받음

(2) 총합 점수를 계산할 변수 `total`을 0으로 초기화 함

(3) 학생들의 점수 배열의 각 원소를 한번씩 순회하며 아래 과정을 반복

- 점수 조회 후 조회한 점수를 `total`에 더함

(4) `total`을 학생수(배열의 크기)로 나누어 반환함

- 장점:
  - 쉽게 이해할 수 있음 (넓은 독자가 이해 가능)
  - 알고리즘의 주요 논리와 개념을 강조할 수 있음
- 단점:
  - 부정확한 해석의 여지가 있어 오해를 불러일으킬 수 있음
  - 자연어를 코드로 번역시 추가 단계가 (많이) 필요함

Example: https://minseokjgit.github.io/papers/PL4XGL_ProSys_slide.pdf

알고리즘 표현 방법 1: 자연어 (Natural Language)

- Example: 입력받은 두 자연수의 최대공약수(GCD)를 구하는 알고리즘을 자연어로 기술하기

알고리즘 표현 방법 2: 프로그래밍 언어 (Programming Language)

- 프로그래밍 언어로 알고리즘을 표현시 가장 자세하고 실행할 수 있는 표현임

```
int MeanMidtermScore(Student* student_array, int size) {
  int total = 0;
  for (int i = 0; i < size; ++i) {
   total += student_array[i].midterm;
  }
  return total / size;
}
```

- 장점:
  - 완전하고 실행 가능한 설명을 제공
  - 즉각적인 테스트 및 검증이 가능함
- 단점:
  - 이해하기 위해 특정 프로그래밍 언어에 대한 지식이 필요
  - 디테일한 정보가 많아 직관적인 동작을 이해하기 어려울 수 있음

Ocaml

```
let mean lst size =
  if size = 0 then
    None
  else
    let total_sum = List.fold_left (+.) 0.0 lst in
    Some (total_sum /. float_of_int size)
```

Example: https://github.com/kupl/PL4XGL/blob/main/GraphClassification/synthesis/generalize.py

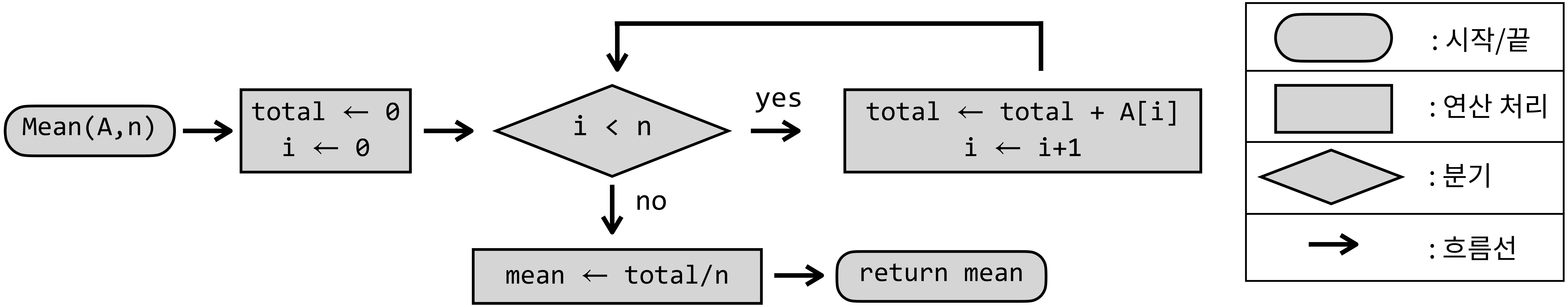
알고리즘 표현 방법 2: 프로그래밍 언어 (Programming Language)

- Example: 입력받은 두 자연수의 최대공약수(GCD)를 구하는 알고리즘을 C언어로 기술하기

```
#include <stdio.h>
int gcd(int a, int b) {




                         ?




}
int main() {
    int num1, num2;
    printf("두 정수를 입력하세요: ");
    scanf("%d %d", &num1, &num2);
    printf("GCD(%d, %d) = %d\n", num1, num2, gcd(num1, num2));
    return 0;
}
```

알고리즘 표현 방법 3: 순서도 (Flowchart)

- 순서도는 그림을 사용하여 알고리즘의 단계와 흐름을 표현함.



- 장점:
  - 그림으로 표현되어 있어 논리의 흐름을 쉽게 따라갈 수 있음
  - 분기점 및 결정 지점을 보여주는 데 효과적
- 단점:
  - 알고리즘이 복잡해질 경우 순서도도 복잡하고 어려워 질 수 있음
  - 순서도를 만들고 수정하는데 많은 시간이 소요됨

알고리즘 표현 방법 3: 순서도 (Flowchart)

- Example: 입력받은 두 자연수의 최대공약수(GCD)를 구하는 알고리즘의 순서도를 기술하기

알고리즘 표현 방법 4: 슈도코드 (Pseudocode)

- 슈도코드는 자연어와 기본 프로그래밍 구조를 결합하여 알고리즘을 설명함.

```
procedure mean(⟨d_1,d_2,…,d_n⟩)
  sum ← 0
  for i ← 1 to n do
    sum ← sum + d_i
  end for
  mean ← sum/n
  return mean
end procedure
```

- 장점:
  - 프로그래머가 아닌 사람도 읽고 이해하기 쉬움
  - 알고리즘에 핵심 논리만을 구조적으로 표현할 수 있음
- 단점:
  - 구현 디테일을 표현하지 못할 수 있음

Example: https://minseokjgit.github.io/papers/PLDI24.pdf

알고리즘 표현 방법 4: 슈도코드 (Pseudocode)

- Example: 입력받은 두 자연수의 최대공약수(GCD)를 구하는 알고리즘의 슈도코드를 기술하기

알고리즘 표현방법

자연어 (Natural Language)

(1) 학생들의 점수의 배열을 입력으로 받음

(2) 총합 점수를 계산할 변수 `total`을 설정함.

(3) 학생들의 점수 배열의 각 원소를 한번씩 순회하며 아래 과정을 반복

- 점수 조회 후 조회한 점수를 `total`에 더함

(4) 합산된 점수를 학생수(배열의 크기)로 나누어 반환함

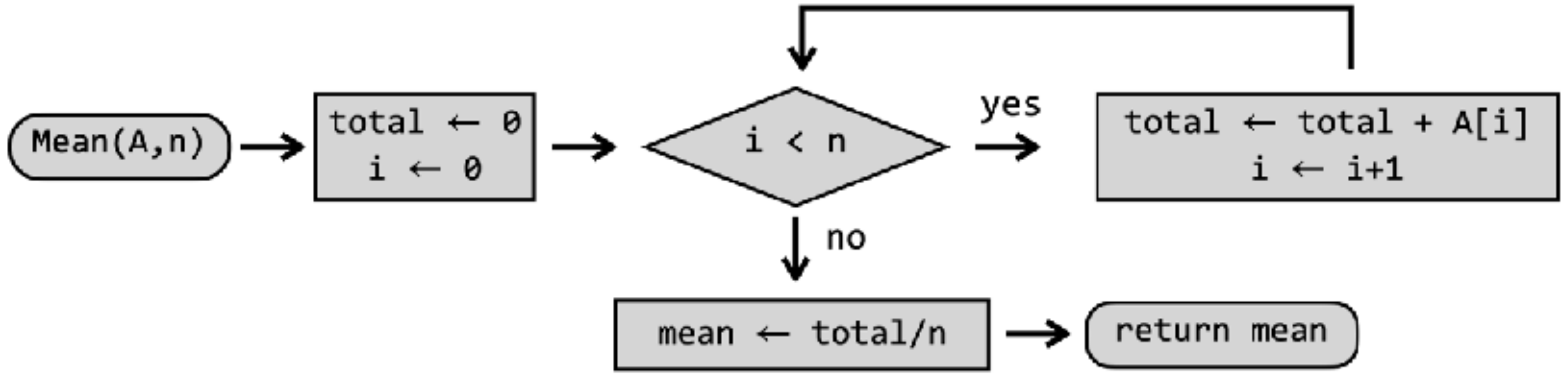
프로그래밍 언어 (Programming Language)

```
int MeanMidtermScore(Student* student_array, int size) {
  int total = 0;
  for (int i = 0; i < size; ++i) {
   total += student_array[i].midterm;
  }
  return total / size;
}
```

슈도코드 (Pseudocode)

```
procedure mean(⟨d_1,d_2,…,d_n⟩)
  sum ← 0
  for i ← 0 to n do
    sum ← sum + d_i
  end for
  mean ← sum/n
  return mean
end procedure
```

순서도 (Flowchart)



문제: 알고리즘의 성능 비교

- 다음 두 최대공약수(GCD)를 구하는 알고리즘의 비용은 각각 얼마인가?

```
procedure Algorithm1(a, b)
  gcd ← 1
  for i = 1 to b do
    if (a % i == 0) and (b % i == 0) then
        gcd ← i
    end if
  end for
  return gcd
end procedure
```

```
procedure Algorithm2(a, b)
  while b ≠ 0 do
    temp ← b
    b ← a%b
    a ← temp
  end while
  return a
end procedure
```

문제: 알고리즘의 성능 비교

- 배열에서는 읽기(read)와 업데이트(update)가 빈번하게 사용됨
    - 예: 중간고사 점수 읽기 + 업데이트하기
- 질문: 배열에서 읽기(read)와 업데이트(update)는 효율적인가?
- 아래의 읽기(read)와 업데이트(update) 알고리즘의 비용(복잡도)은 얼마일까?
    - 읽기(read operation):

```
int read(int arr[], int index) {
  return arr[index];
}
```

    - 업데이트(update operation):

```
void update(int arr[], int index, int newValue) {
        arr[index] = newValue;
}
```

문제: 알고리즘의 성능 비교

- 실제 실행 시간으로 비교할 경우

```c
#include <stdio.h>
#include <time.h>


int algorithm1(int a, int b) {
    int gcd = 1;
    for (int i = 1; i <= b ; ++i){
        if ((a%i) == 0 && (b%i) == 0){
            gcd = i;
        }
    }
    return gcd;
}

int algorithm2(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

```c
int main() {
    int num1, num2;

    printf("두 정수를 입력하세요: ");
    scanf("%d %d", &num1, &num2);
    double cost1, cost2;
    clock_t start1, start2, finish1, finish2;

    start1 = clock();
    printf("GCD = %d\n", algorithm1(num1, num2));
    finish1 = clock();
    cost1 = (double)(finish1 - start1)/ CLOCKS_PER_SEC;
    printf("Cost of algorithm 1 : %f\n", cost1);

    start2 = clock();
    printf("GCD = %d\n", algorithm2(num1, num2));
    finish2 = clock();
    cost2 = (double)(finish2 - start2)/ CLOCKS_PER_SEC;
    printf("Cost of algorithm 2 : %f\n", cost2);

    return 0;
}
```

문제: 알고리즘의 성능 비교

- 실제 실행 시간으로 비교할 경우 문제점
  - 실제로 구현해야 함
  - 동일한 조건의 하드웨어를 사용해서 실행시간을 측정해야함
  - 사용한 소프트웨어 환경도 동일해야 함 (예: 같은 프로그래밍 언어로 구현해야 함)
  - 성능 비교에 사용한 데이터가 아닌 다른 데이터에 대해서는 다른 결과가 나올 수 있음

알고리즘의 복잡도 분석

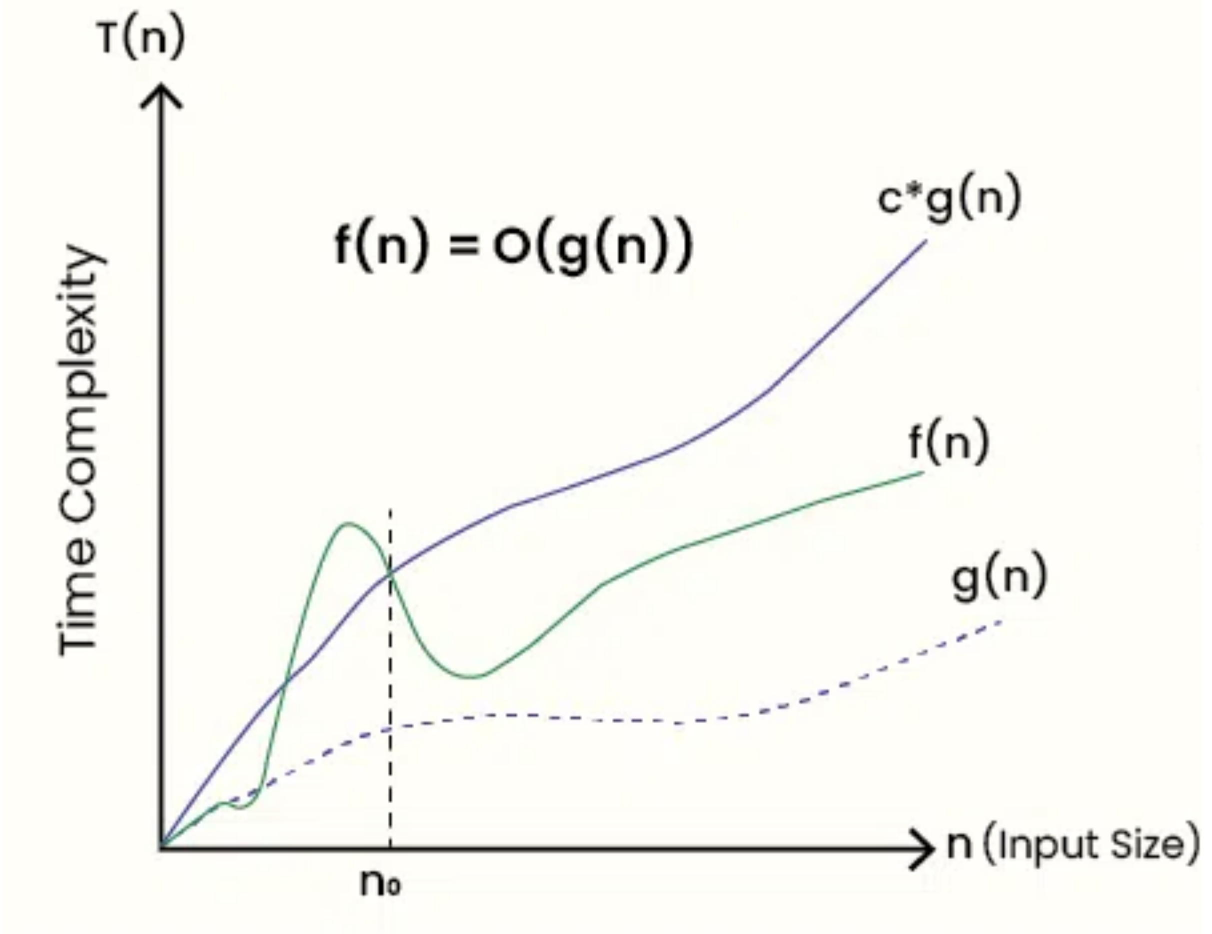
- **알고리즘의 복잡도 분석**은 주어진 입력(크기 n)에 대해 알고리즘이 완료되는 데 걸리는 비용(시간과 공간)을 측정하는 방법임
  - 사용하는 기계, 프로그래밍 언어, 컴파일러에 독립적으로 측정함
- 왜 **알고리즘의 복잡도 분석**을 해야할까?
  - 주어진 문제에 적합한 알고리즘(또는 자료구조)이 무엇인지 알아내기 위해
- **알고리즘의 복잡도 분석을 위한** 세 가지 점근 표기법 (asymptotic notations)
  - Big-O (O) : 알고리즘이 최악으로 동작하는 경우(worst-case)에 해당하는 비용을 표현
  - Big Omega (Ω) : 알고리즘이 최선으로 동작하는 경우(best-case)에 해당하는 비용을 표현
  - Theta notation (Θ) : 알고리즘이 평균적인 비용을 표현

https://www.scholarhat.com/tutorial/datastructures/complexity-analysis-of-data-structures-and-algorithms

Big-O 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = O(g(n))$ *or* $f(n) \in O(g(n))$ *if there exist positive constant* c *and* n_0 *such that*

$$\forall n \geq n_0 \,.\, 0 \leq f(n) \leq c \cdot g(n)$$



https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/?ref=gcse_ind

Big-O 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = O(g(n))$ *or* $f(n) \in O(g(n))$ *if there exist positive constant* c *and* n_0 *such that*

$$\forall n \geq n_0 \,.\, 0 \leq f(n) \leq c \cdot g(n)$$

- **Examples.**
  - $T(n) = 5 \implies T(n) \in O(1)$ ($c = 5$ and $n_0 = 1$)
  - $T(n) = 2n + 1 \implies T(n) \in O(n)$ ($c = 3$ and $n_0 = 1$)
  - $T(n) = 3n^2 + n + 100 \implies T(n) \in O(n^2)$ ($c = 4$ and $n_0 = 11$)
  - $T(n) = 2^n + n^2 \implies T(n) \in O(2^n)$ ($c = 2$ and $n_0 = 4$)
  - $T(n) = n \cdot log\ n + n \implies T(n) \in O(n \cdot log\ n)$ ($c = 2$ and $n_0 = 4$)
- **시간 복잡도 비교:**

$$O(1) < O(n) < O(n \cdot log\ n) < O(n^2) < O(2^n)$$

Big-O 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = O(g(n))$ *or* $f(n) \in O(g(n))$ *if there exist positive constant* c *and* n_0 *such that*

$$\forall n \geq n_0 \,.\, 0 \leq f(n) \leq c \cdot g(n)$$

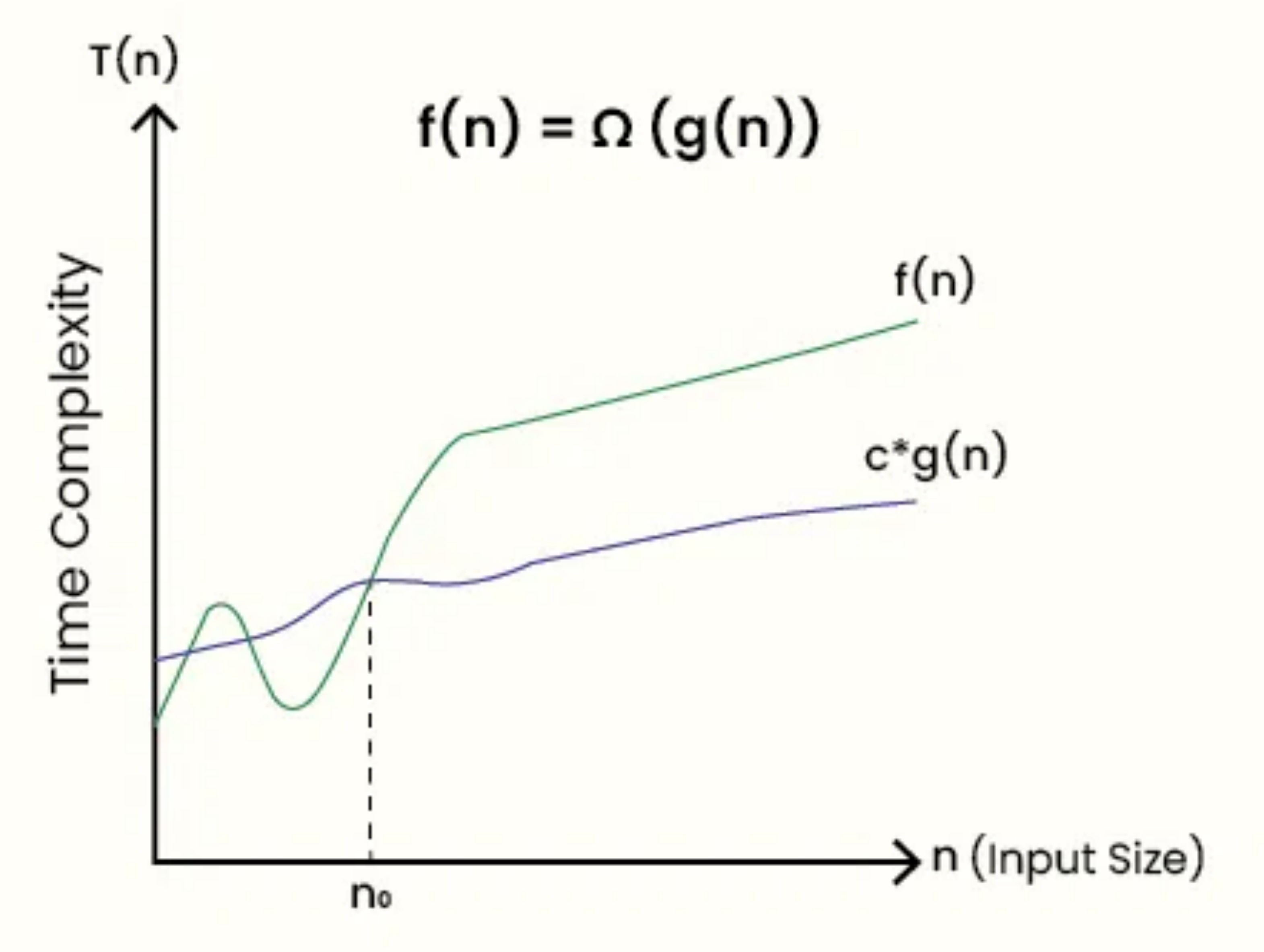
- **Examples.**
  - $T(n) \in O(n) \implies T(n) \in O(n^2)$
  - $T(n) \in O(n) \implies T(n) \in O(2^n)$
- **시간 복잡도 비교:**

$$O(1) < O(n) < O(n \cdot log\ n) < O(n^2) < O(2^n)$$

Big-Omega (Ω) 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = \Omega(g(n))$ *or* $f(n) \in \Omega(g(n))$ *if there exist positive constants c and* n_0 *such that:*

$$\forall n \geq n_0 .\ \ f(n) \geq c \cdot g(n)$$



https://www.geeksforgeeks.org/analysis-of-algorithms-big-omega-notation/

Big-Omega (Ω) 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = \Omega(g(n))$ *or* $f(n) \in \Omega(g(n))$ *if there exist positive constants c and* n_0 *such that:*

$$\forall n \geq n_0 .\ f(n) \geq c \cdot g(n)$$

- **Examples.**
  - $T(n) = 5 \implies T(n) \in \Omega(1)$ ($c = ?$ and $n_0 = ?$)
  - $T(n) = 2n + 1 \implies T(n) \in \Omega(n)$ ($c = ?$ and $n_0 = ?$)
  - $T(n) = 3n^2 + n + 100 \implies T(n) \in \Omega(n^2)$ ($c = ?$ and $n_0 = ?$)
  - $T(n) = 2^n + n^2 \implies T(n) \in \Omega(2^n)$ ($c = ?$ and $n_0 = ?$)
  - $T(n) = n \cdot log\ n + n \implies T(n) \in \Omega(n \cdot log\ n)$ ($c = ?$ and $n_0 = ?$)

Big-Omega (Ω) 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = \Omega(g(n))$ *or* $f(n) \in \Omega(g(n))$ *if there exist positive constants c and* n_0 *such that:*

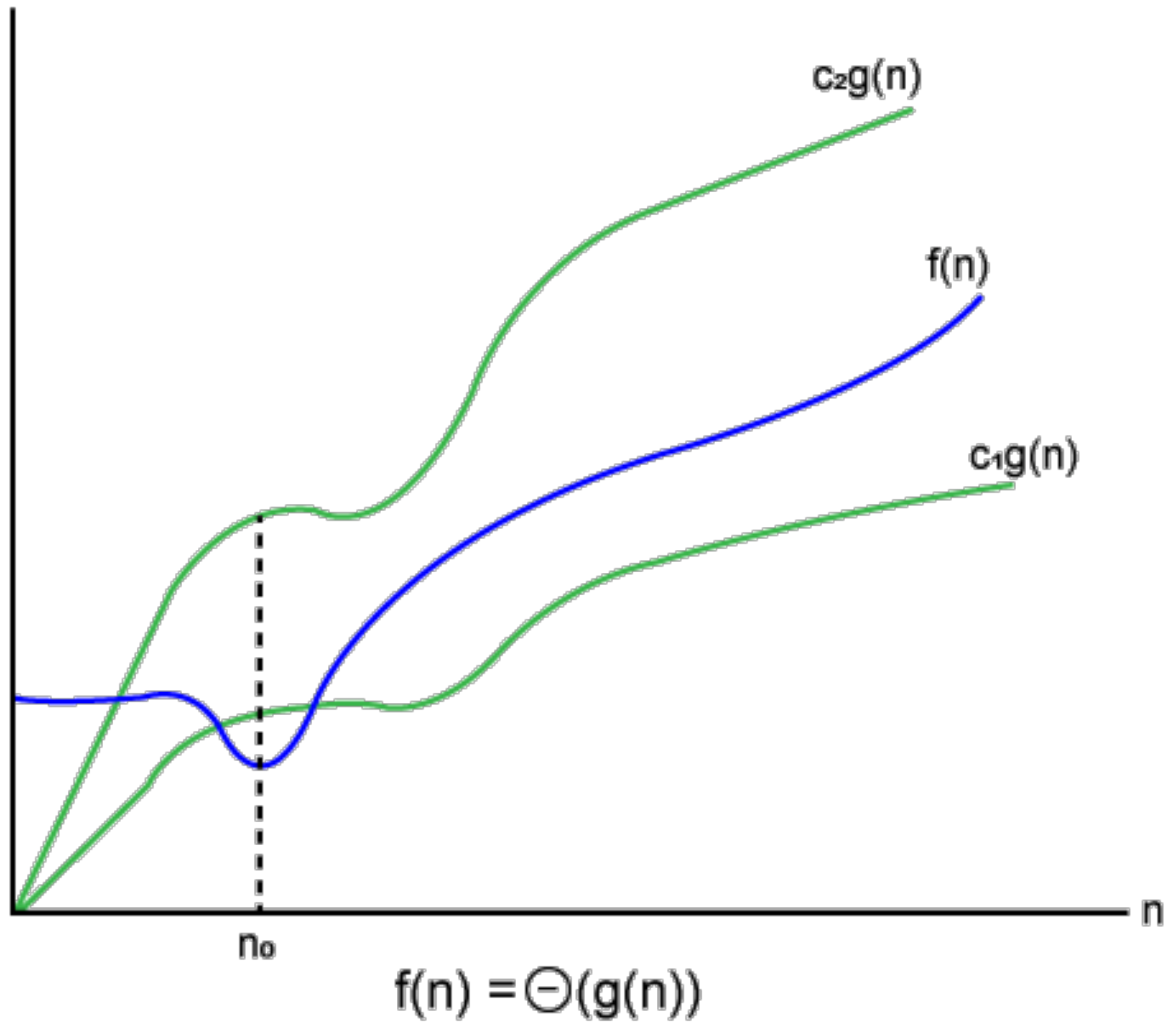
$$\forall n \geq n_0 .\ f(n) \geq c \cdot g(n)$$

- **Examples.**
    - $T(n) \in \Omega(n^2) \implies T(n) \in \Omega(n)$
    - $T(n) \in \Omega(2^n) \implies T(n) \in \Omega(n)$

Big-Theta (Θ) 표기법

Definition. *For two functions f(n) and g(n), we say that $f(n) = \Theta(g(n))$ or $f(n) \in \Theta(g(n))$ if there exist positive constants c_1, c_2, and n_0 such that:*

$$\forall n \geq n_0 .\ \ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



https://www.javatpoint.com/data-structure-asymptotic-analysis

Big-Theta (Θ) 표기법

Definition. *For two functions f(n) and g(n), we say that* $f(n) = \Theta(g(n))$ *or* $f(n) \in \Theta(g(n))$ *if there exist positive constants* c_1, c_2, *and* n_0 *such that:*

$$\forall n \geq n_0 .\ \ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- **Examples.**
  - $T(n) = 5 \implies T(n) \in \Theta(1)$ $\quad (c_1 = ?, c_2 = ?$, and $n_0 = ?)$
  - $T(n) = 2n + 1 \implies T(n) \in \Theta(n)$ $\quad (c_1 = ?, c_2 = ?$, and $n_0 = ?)$
  - $T(n) = 3n^2 + n + 100 \implies T(n) \in \Theta(n^2)$ $\quad (c_1 = ?, c_2 = ?$, and $n_0 = ?)$
  - $T(n) = 2^n + n^2 \implies T(n) \in \Theta(2^n)$ $\quad (c_1 = ?, c_2 = ?$, and $n_0 = ?)$
  - $T(n) = n \cdot log\ n + n \implies T(n) \in \Theta(n \cdot log\ n)$ $\quad (c_1 = ?, c_2 = ?$, and $n_0 = ?)$

Example

- 배열에서 읽기(read)와 업데이트(update) 알고리즘의 비용(복잡도)은 얼마일까?

	Algorithm	Time complexity
read	**procedure** read($\langle d_1, d_2, \ldots, d_n \rangle$, idx) **return** d_{idx} **end procedure**	
update	**procedure** update($\langle d_1, d_2, \ldots, d_n \rangle$, idx, val) $d_{idx} \leftarrow$ val **return** $\langle d_1, d_2, \ldots, d_n \rangle$ **end procedure**	

Example

- 아래 알고리즘의 복잡도 (big-O)는 얼마인가?

procedure search($\langle d_1, d_2, \ldots, d_n \rangle$, val)
  idx ← 0
  for i ←1 to n **do**
    if d_i = val **then**
      return i
    end if
  return -1
end procedure

마무리 (Wrap-up)

- 문제: 주어진 알고리즘 및 자료구조가 얼마나 문제에 적합한지 분석해야 함
- 해결책: 알고리즘의 복잡도 분석
- **알고리즘의 복잡도 분석을 위한** 세 가지 점근 표기법 (asymptotic notations):
    - Big-O (O) : 알고리즘이 최악으로 동작하는 경우(worst-case)에 해당하는 비용을 표현
    - Big Omega (Ω) : 알고리즘이 최선으로 동작하는 경우(best-case)에 해당하는 비용을 표현
    - Theta notation (Θ) : 알고리즘이 평균적인 비용을 표현