제2장 알고리즘의 기초 개념

과 목 명　　정 보 처 리 알 고 리 즘

담당교수　김　성　훈

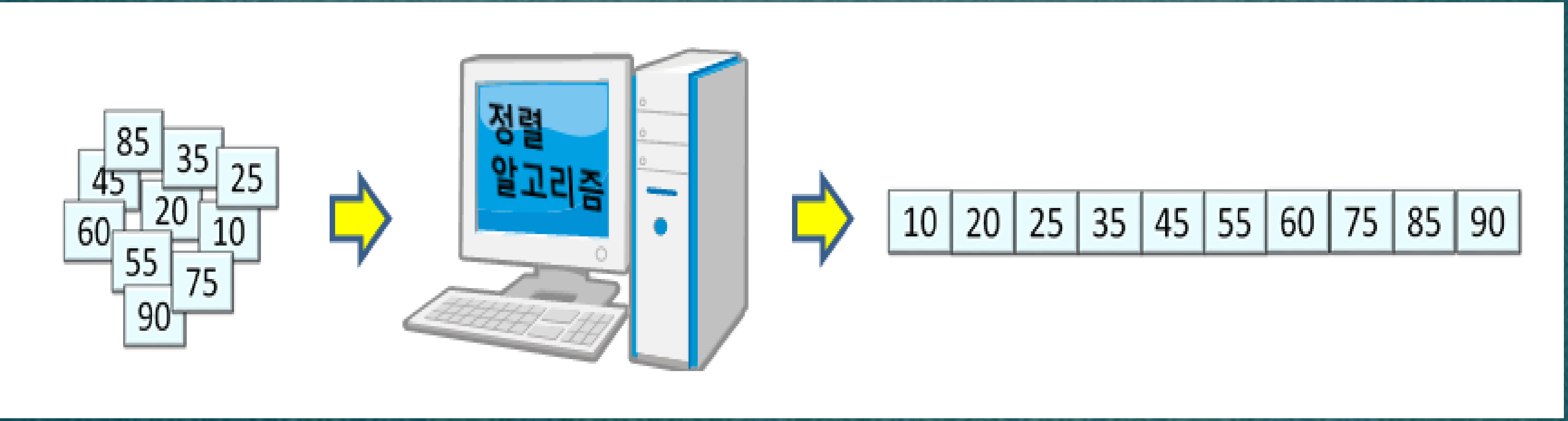
경 북 대 학 교　과 학 기 술 대 학　소 프 트 웨 어 학 과

이 장에서 배울 내용

1. 알고리즘이란?
2. 최초의 알고리즘, 유클리드호젯법
3. 알고리즘의 표현방법
4. 알고리즘의 분류
5. 알고리즘의 효율성 표현
6. 복잡도의 점근적 표기
7. 왜 효율적인 알고리즘이 필요한가?

2.1 알고리즘이란

- 알고리즘은 문제를 해결하는 단계적 절차 또는 방법이다.
- 주어지는 문제는 컴퓨터를 이용하여 해결할 수 있어야 한다.
- 알고리즘에는 입력이 주어지고,
  알고리즘은 수행한 결과인 해 (또는 답)를 출력한다.



알고리즘의 일반적인 특성

- 정확성:
  - 알고리즘은 주어진 입력에 대해 올바른 해를 주어여야 한다.
- 수행성:
  - 알고리즘의 각 단계는 컴퓨터에서 수행 가능하여야 한다.
- 유한성:
  - 알고리즘은 일정한 시간 내에 종료되어야 한다.
- 효율성:
  - 알고리즘은 효율적일수록 그 가치가 높아진다.

2.2 최초의 알고리즘

- 가장 오래된 알고리즘:
  기원전 300년경 유클리드 (Euclid)의 최대공약수 알고리즘

  - 최대공약수(GCD; Greatest Common Devisor)는
    2개 이상의 자연수의 공약수들 중에서 가장 큰 수

  - 유클리드는 2개 자연수의 최대공약수는
    "큰 수에서 작은 수를 뺀 수(A-B)와
    작은 수와의 최대공약수가 같다" 라는 성질을 이용했다.

    GCD(A, B) = GCD(A–B, B), 단 A>B

Gcd(24, 14)

= Gcd(24-14, 14)

= Gcd(10, 14) = Gcd(14-10, 10)

= Gcd(4, 10) = Gcd(10-4, 4)

= Gcd(6, 4) = Gcd(6-4, 4)

= Gcd(2, 4) = Gcd(4-2, 2)

= Gcd(2, 2) = Gcd(2-2, 2)

= Gcd(2, 0)

= 2

유클리드의 최대공약수 알고리즘

Euclid(a, b)
입력: 정수 a, b; 단, $a \geq b \geq 0$
출력: 최대공약수(a, b)

1. if (b=0) return a
2. return Euclid(b, a mod b)

최대공약수(24, 14)

- Line 1: b=14이므로 if-조건이 '거짓'
- Line 2: Euclid(14, 24 mod 14) = Euclid(14, 10) 호출
- Line 1: b=10이므로 if-조건이 '거짓'
- Line 2: Euclid(10, 14 mod 10) = Euclid(10, 4) 호출
- Line 1: b=4이므로 if-조건이 '거짓'
- Line 2: Euclid(4, 10 mod 4) = Euclid(4, 2) 호출
- Line 1: b=2이므로 if-조건이 '거짓'
- Line 2: Euclid(2, 4 mod 2) = Euclid(2, 0) 호출
- Line 1: b=0이므로 if-조건이 '참'이 되어 a=2를 최종적으로 리턴

2.3 알고리즘의 표현 방법

- 알고리즘의 형태는 단계별 절차이므로, 마치 요리책의 요리를 만드는 절차와 유사
- 알고리즘의 각 단계는 보통 말로 서술할 수 있으며, 컴퓨터 프로그래밍 언어로만 표현할 필요는 없다.
- 일반적으로 알고리즘은 프로그래밍 언어와 유사한 의사 코드 (pseudo code)로 표현

최대 숫자 찾기 문제를 위한 알고리즘

보통 말로 표현된 알고리즘:

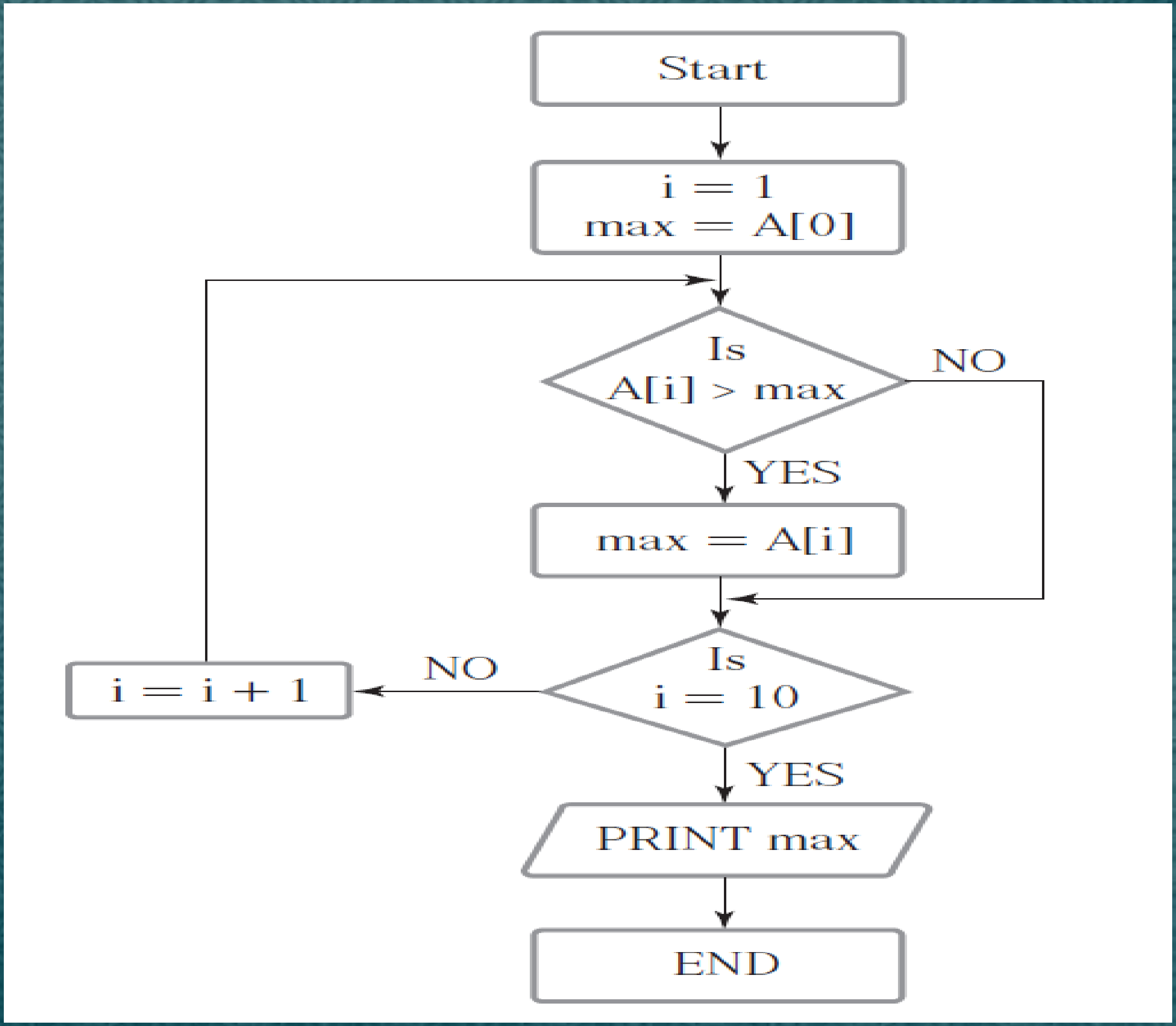
1. 첫 카드의 숫자를 읽고 머릿속에 기억해 둔다.
2. 다음 카드의 숫자를 읽고, 그 숫자를 머릿속의 숫자와 비교한다.
3. 비교 후 큰 숫자를 머릿속에 기억해 둔다.
4. 다음에 읽을 카드가 남아있으면 line 2로 간다.
5. 머릿속에 기억된 숫자가 최대 숫자이다.

의사 코드로 표현된 알고리즘:

배열 A에 입력이 10개의 숫자가 있다고 가정

1. max = A[0]
2. for i = 1 to 9
3. if (A[i] > max) max = A[i]
4. return max

플로우차트 (flow chart) 형태



BREAK TIME

2.4 알고리즘의 분류

- 문제의 해결 방식에 따른 분류:
  - 분할 정복 (Divide-and-Conquer) 알고리즘 (제3장)
  - 그리디 (Greedy) 알고리즘 (제4장)
  - 동적 계획 (Dynamic Programming) 알고리즘 (제5장)
  - 근사 (Approximation) 알고리즘 (제8장)
  - 백트래킹 (Backtracking) 기법 (제9장)
  - 분기 한정 (Branch-and-Bound) 기법 (제9장)

2.4 알고리즘의 분류(계속)

- 문제에 기반한 분류:
  - 정렬 알고리즘 (제6장)
  - 그래프 알고리즘
  - 기하 알고리즘
- 특정 환경에 따른 분류:
  - 병렬 (Parallel) 알고리즘
  - 분산 (Distributed) 알고리즘
  - 양자 (Quantum) 알고리즘

2.4 알고리즘의 분류(계속)

- 기타 알고리즘들:
  - 확률 개념이 사용되는 랜덤 (Random) 알고리즘
  - 유전자 (Genetic) 알고리즘 (제9장)

2.5 알고리즘의 효율성 표현

- 알고리즘의 효율성은 알고리즘의 수행 시간 또는 알고리즘이 수행하는 동안 사용되는 메모리 공간의 크기로 나타낼 수 있다.
- 이들을 각각 시간복잡도 (time complexity), 공간복잡도 (space complexity)라고 한다.
- 일반적으로 알고리즘들을 비교할 때에는 시간복잡도가 주로 사용된다

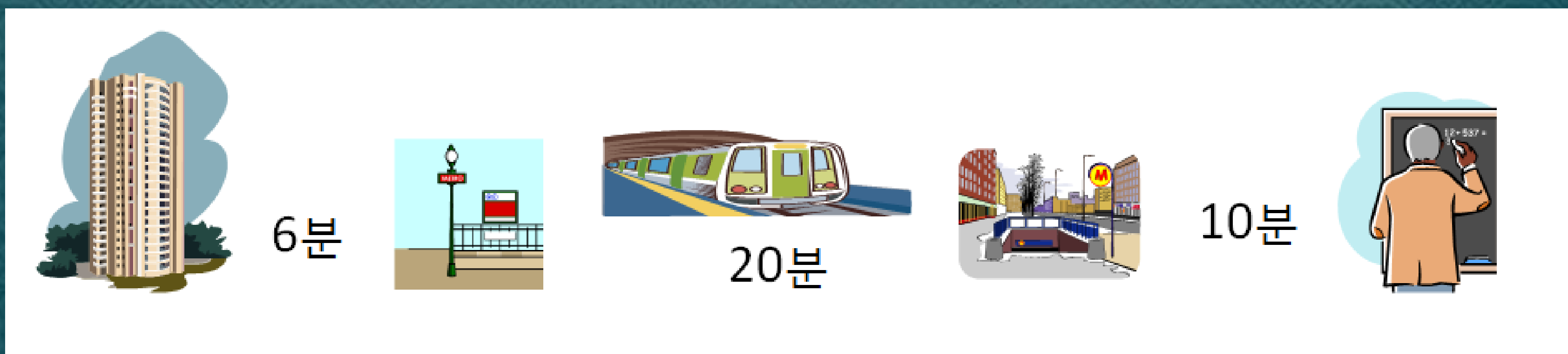
시간복잡도

- 시간복잡도는 알고리즘이 수행하는 기본적인 연산 횟수를 입력 크기에 대한 함수로 표현
  - 예: 10장의 숫자 카드 중에서 최대 숫자 찾기 순차탐색으로 찾는 경우에 숫자 비교가 기본적인 연산이고, 총 비교 횟수는 9이다.
  - n장의 카드가 있다면, (n-1)번의 비교 수행: 시간복잡도는 (n-1)

알고리즘의 복잡도를 표현 방법

- 알고리즘의 복잡도를 표현하는 데는 다음과 같은 분석 방법들이 있다.
  - 최악 경우 분석 (worst case analysis)
  - 평균 경우 분석 (average case analysis)
  - 최선 경우 분석 (best case analysis)

최선 경우



최악 경우

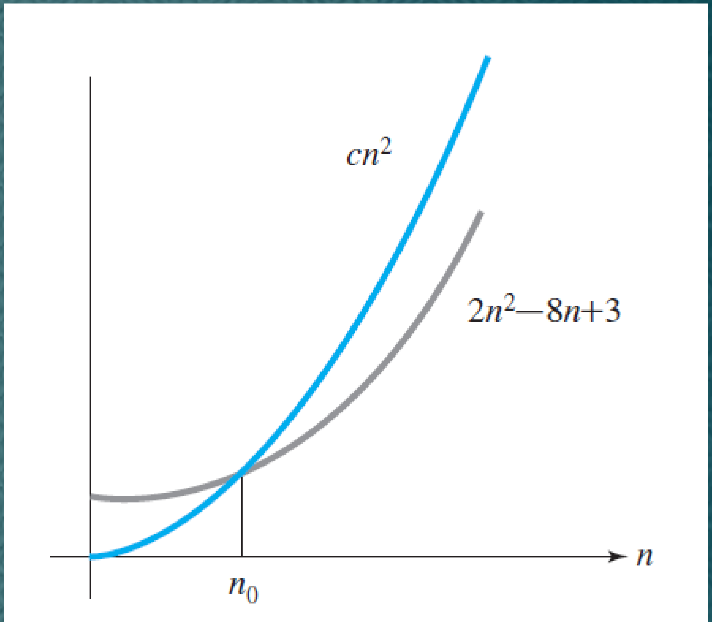
평균 경우

2.6 복잡도의 점근적 표기

- 시간 (또는 공간)복잡도는 입력 크기에 대한 함수로 표기하는데, 이 함수는 주로 여러 개의 항을 가지는 다항식이다.
- 이를 단순한 함수로 표현하기 위해 점근적 표기(Asymptotic Notation)를 사용한다.
- 입력 크기 n이 무한대로 커질 때의 복잡도를 간단히 표현하기 위해 사용하는 표기법이다.
- O(Big-Oh) -표기
- Ω(Big-Omega) -표기
- Θ(Theta) -표기

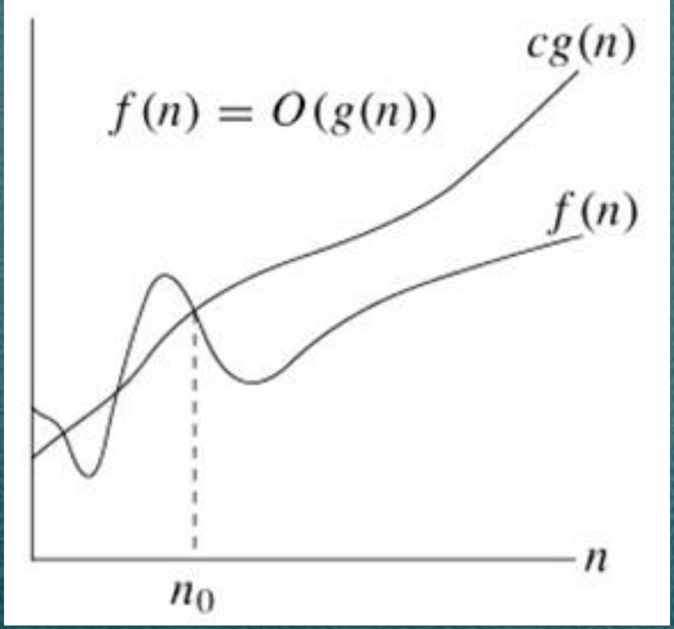
O(Big-Oh)-표기

- O-표기는 복잡도의 점근적 상한을 나타낸다.
- 복잡도가 $f(n) = 2n^2-8n+3$ 이라면, $f(n)$의 O-표기는 $O(n^2)$이다. 먼저 $f(n)$의 단순화된 표현은 n^2이다.
- 단순화된 함수 n^2에 임의의 상수 c를 곱한 cn^2이 n이 증가함에 따라 $f(n)$의 상한이 된다. 단, $c>0$.



O(Big-Oh)-표기

- 복잡도 f(n)과 O-표기를 그래프로 나타내고 있다.
- n이 증가함에 따라 O(g(n))이 점근적 상한이라는 것 (즉, g(n)이 n_0보다 큰 모든 n에 대해서 항상 f(n)보다 크다는 것)을 보여 준다.

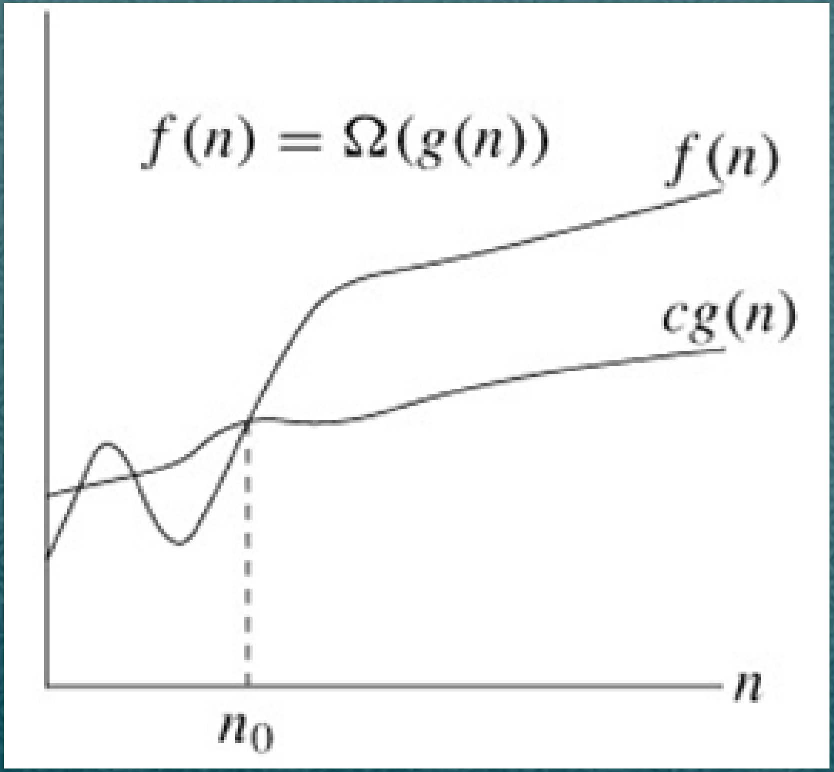


Ω(Big-Omega)-표기

- 복잡도의 점근적 하한을 의미한다.
- $f(n) = 2n^2-8n+3$의 Ω-표기는 $\Omega(n^2)$이다.
- $f(n)=\Omega(n^2)$은 "n이 증가함에 따라 $2n^2-8n+3$이 cn^2보다 작을 수 없다"라는 의미이다. 이때 상수 c=1로 놓으면 된다.
- O-표기 때와 마찬가지로, Ω-표기도 복잡도 다항식의 최고차항만 계수 없이 취하면 된다.

Ω(Big-Omega)-표기

- 복잡도 f(n)과 Ω-표기를 그래프로 나타낸 것인데, n이 증가함에 따라 Ω(g(n))이 점근적 하한이라는 것 (즉, g(n)이 n_0보다 큰 모든 n에 대해서 항상 f(n)보다 작다는 것)을 보여준다.



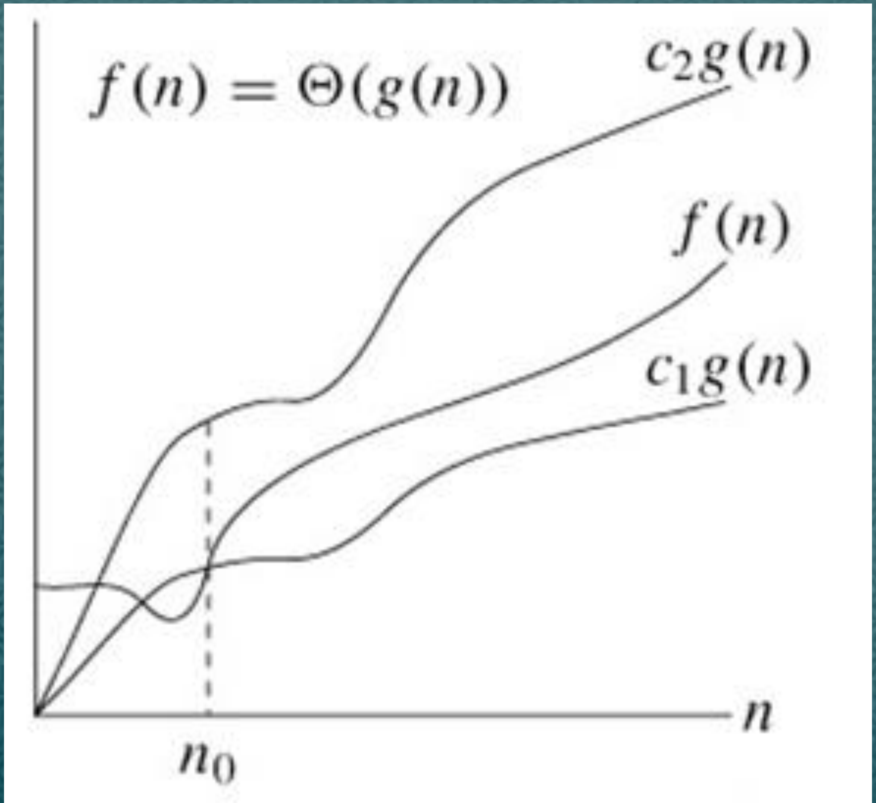
BREAK TIME

Θ(Theta)-표기

- O-표기와 Ω-표기가 같은 경우에 사용한다.
- $f(n) = 2n^2+10n+3 = O(n^2) = \Omega(n^2)$이므로, $f(n)=\Theta(n^2)$이다.
- "f(n)은 n이 증가함에 따라 n^2과 동일한 증가율을 가진다" 라는 의미이다.
- $f(n)\neq\Theta(n)$, $f(n)\neq\Theta(n\log n)$, $f(n)\neq\Theta(n^3)$, $f(n)\neq\Theta(2^n)$이다.

Θ(Theta)-표기

- 복잡도 f(n)과 Θ-표기를 그래프로 n_0보다 큰 모든 n에 대해서 Θ-표기가 상한과 하한 동시에 만족한다는 것을 보여준다.



자주 사용하는 O-표기

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

O-표기의 포함 관계

$14, \ 1000$ — $O(1)$

$O(\log n)$

$3\log n+6$ — $O(n)$

$O(n^2)$

$5n+4$ — $O(n^3), \ O(n^k)$

$O(2^n)$

$2n^2+10n+3$

$n^3+n^2-10n+3$

2^n

2.7 왜 효율적인 알고리즘이 필요한가?

- 10억 개의 숫자를 정렬하는데 PC에서 $O(n^2)$ 알고리즘은 300여년이 걸리는 반면에 O(nlogn) 알고리즘은 5분 만에 정렬한다.

$O(n^2)$	1,000	1백만	10억
PC	< 1초	2시간	300년
슈퍼컴	< 1초	1초	1주일

O(nlogn)	1,000	1백만	10억
PC	< 1초	<1초	5분
슈퍼컴	< 1초	<1초	<1초

2.7 왜 효율적인 알고리즘이 필요한가?

- 효율적인 알고리즘은 슈퍼컴퓨터보다 더 큰 가치가 있다.
- 값 비싼 H/W의 기술 개발보다 효율적인 알고리즘 개발이 훨씬 더 경제적이다.

요약

- 알고리즘이란 문제를 해결하는 단계적 절차 또는 방법이다.
- 알고리즘의 일반적인 특성
  - 정확성: 주어진 입력에 대해 올바른 해를 주어여야 한다.
  - 수행성: 각 단계는 컴퓨터에서 수행 가능하여야 한다.
  - 유한성: 일정한 시간 내에 종료되어야 한다.
  - 효율성: 효율적일수록 그 가치가 높다.
- 알고리즘은 대부분 의사 코드 (pseudo code) 형태로 표현된다.

요약(계속)

- 알고리즘의 효율성은 주로 시간복잡도 (Time Complexity)가 사용된다.
- 시간복잡도는 알고리즘이 수행하는 기본적인 연산 횟수를 입력 크기에 대한 함수로 표현한다.
- 알고리즘의 복잡도 표현 방법:
  - 최악 경우 분석 (worst case analysis)
  - 평균 경우 분석 (average case analysis)
  - 최선 경우 분석 (best case analysis)
- 점근적 표기 (Asymptotic Notation): 입력 크기 n이 무한대로 커질 때의 복잡도를 간단히 표현하기 위해 사용하는 표기법
  - O-(Big-Oh) 표기: 점근적 상한
  - Ω-(Big-Omega) 표기: 점근적 하한
  - Θ-(Theta) 표기: 동일한 증가율

Q&A