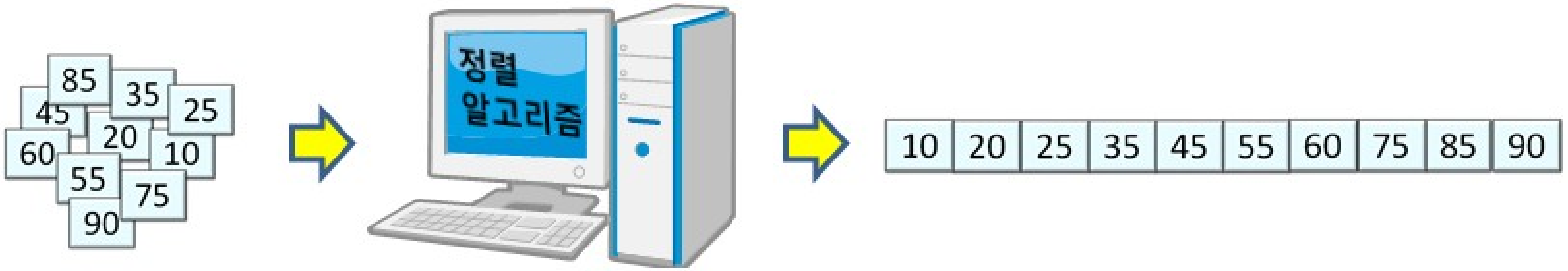
Chapter 2
알고리즘을 배우기 위한 준비

차례

2.1 알고리즘이란

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



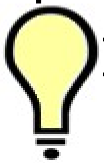
알고리즘의 일반적 특성

- 정확성
  - 알고리즘은 주어진 입력에 대해 올바른 해를 주어야(랜덤 알고리즘은 예외)
- 수행성
  - 알고리즘의 각 단계는 컴퓨터에서 수행 가능해야
- 유한성
  - 알고리즘은 유한 시간 내에 종료되어야
- 효율성
  - 알고리즘은 효율적일수록 그 가치가 높아진다.

2.2 최초의 알고리즘

- 유클리드(Euclid)의 최대공약수 알고리즘
  - 기원전 300년경에 만들어진 가장 오래된 알고리즘
  - 최대공약수란 2개의 자연수의 공약수들 중에서 가장 큰 수

2개의 자연수의 최대공약수는 큰 수에서 작은 수를 뺀 수와 작은 수와의 최대공약수와 같다.



최대공약수(24, 14)

최대공약수(24, 14)

= 최대공약수(24-14, 14) = 최대공약수(10, 14)

= 최대공약수(14-10, 10) = 최대공약수(4, 10)

= 최대공약수(10-4, 4) = 최대공약수(6, 4)

= 최대공약수(6-4, 4)= 최대공약수(2, 4)

= 최대공약수(4-2, 2)= 최대공약수(2, 2)

= 최대공약수(2-2, 2)= 최대공약수(0, 2)

= 2

- 유클리드의 최대공약수 알고리즘에서 뺄셈 대신에 나눗셈을 사용하면 빠르게 해를 찾는다.

알고리즘

```
Euclid(a, b)
입력: 정수 a, b;  단, a≥b≥0
출력: 최대공약수(a, b)

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

유클리드의 최대공약수 알고리즘

```
Euclid(a, b)
입력: 정수 a, b;  단, a≥b≥0
출력: 최대공약수(a, b)
1.  if b==0 return a
2.  return Euclid(b, a mod b)
```

최대공약수(24, 14)

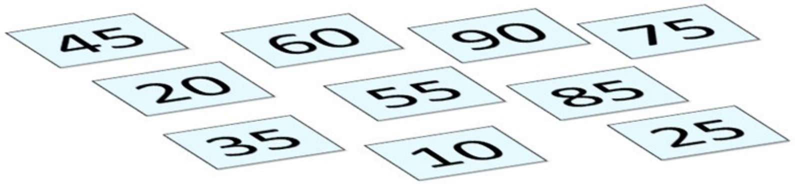
- Line 1: b==14이므로 if-조건이 '거짓'
- Line 2: Euclid(14, 24 mod 14) = Euclid(14, 10) 호출
- Line 1: b==10이므로 if-조건이 '거짓'
- Line 2: Euclid(10, 14 mod 10) = Euclid(10, 4) 호출
- Line 1: b==4이므로 if-조건이 '거짓'
- Line 2: Euclid(4, 10 mod 4) = Euclid(4, 2) 호출
- Line 1: b==2이므로 if-조건이 '거짓'
- Line 2: Euclid(2, 4 mod 2) = Euclid(2, 0) 호출
- Line 1: b==0이므로 if-조건이 '참'이 되어 a=2를 최종 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 없음
- 일반적으로 알고리즘은 프로그래밍 언어와 유사한 의사 코드(pseudo code)로 표현

최대 숫자 찾기 문제를 위한 알고리즘

- 최대 숫자 찾기



  – 카드의 숫자를 하나씩 비교하면서 본 숫자들 중에서 가장 큰 숫자를 기억해가며 찾는다.
  – 마지막 카드의 숫자를 본 후에, 머릿속에 기억된 가장 큰 숫자가 적힌 카드를 바닥에서 집어 든다.

보통 말로 표현된 알고리즘

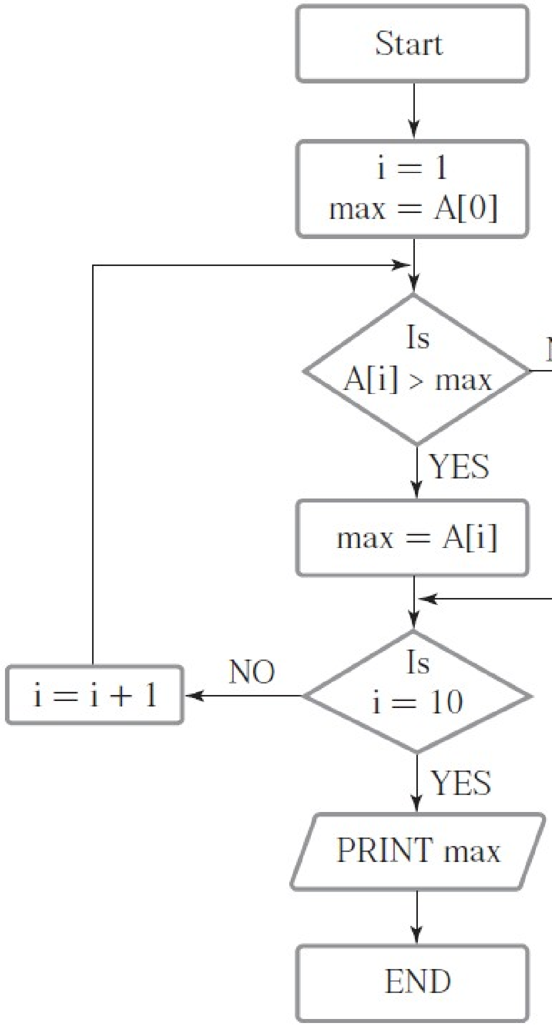
1. 첫 카드의 숫자를 읽고 머릿속에 기억해 둔다.
2. 다음 카드의 숫자를 읽고, 그 숫자를 머릿속의 숫자와 비교한다.
3. 비교 후 큰 숫자를 머릿속에 기억해 둔다.
4. 다음에 읽을 카드가 남아있으면 line 2로 간다.
5. 머릿속에 기억된 숫자가 최대 숫자이다.

의사 코드로 표현된 알고리즘

배열 A에 10개의 숫자가 있다면

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

플로우 차트(flow chart)



2.4 알고리즘의 분류

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

알고리즘의 분류

- 문제에 기반한 분류
  - 정렬 알고리즘
  - 그래프 알고리즘
  - 기하 알고리즘

- 특정 환경에 따른 분류
  - 병렬(Parallel) 알고리즘
  - 분산(Distributed) 알고리즘
  - 양자(Quantum) 알고리즘

- 기타 알고리즘

2.5 알고리즘의 효율성 표현

- 알고리즘의 효율성
  - 알고리즘의 수행 시간 또는 알고리즘이 수행하는 동안 사용되는 메모리 크기로 나타낼 수 있다.
  - 시간 복잡도(time complexity), 공간 복잡도(space complexity)
  - 일반적으로 알고리즘들을 비교할 때에는 시간 복잡도가 주로 사용됨

시간 복잡도

- 시간 복잡도는 알고리즘이 실행되는 동안에 사용된 기본적인 연산 횟수를 입력 크기의 함수로 나타낸다.
  - 기본 연산(Elementary Operation)이란 데이터 간 크기 비교, 데이터 읽기, 갱신, 숫자 계산 등과 같은 단순한 연산을 의미

- [예] 10장의 숫자 카드들 중에서 최대 숫자 찾기
  - 순차 탐색으로 찾는 경우에 숫자 비교가 기본적인 연산이고, 총 비교 횟수는 9번
  - n장의 카드가 있다면, (n-1)번의 비교 수행으로 시간 복잡도는 (n-1)

알고리즘의 복잡도 표현 방법

- 최악 경우 분석(Worst-case Analysis)
  - ‘어떤 입력이 주어지더라도 알고리즘의 수행시간이 얼마 이상은 넘지 않는다‘라는 상한(Upper Bound)의 의미
- 평균 경우 분석(Average-case Analysis)
  - 입력의 확률 분포를 가정하여 분석하는데, 일반적으로 균등 분포(Uniform Distribution)를 가정
- 최선 경우 분석(Best-case Analysis)
  - 가장 빠른 수행 시간을 분석하며, 최적(Optimal) 알고리즘을 찾는데 활용

알고리즘의 복잡도 표현 방법

- 상각 분석(Amortized Analysis)
  - 일련의 연산을 수행하여 총 수행 시간을 합하고 이를 연산 횟수로 나누어 수행 시간을 분석
  - [조건] 알고리즘에서 적어도 두 종류의 연산이 수행되고, 그 중 하나는 수행 시간이 길고, 다른 하나는 짧으며, 수행 시간이 짧은 연산은 많이 수행되고 수행 시간이 긴 연산은 적게 수행되어야 상각 분석이 의미를 갖는다.
  - amortize는 '분할 상환하다'라는 뜻을 가짐. 그리고 상각(償却)은 '보상하여 갚아주다'라는 뜻
- 일반적으로 알고리즘의 수행 시간은 최악 경우 분석으로 표현

등교 시간 분석

- 집에서 지하철역까지 6분, 지하철을 타면 학교까지 20분, 강의실까지 걸어서 10분 걸린다.

- 최선 경우
  - 집을 나와서 6분 후 지하철역에 도착하고, 운이 좋게 바로 열차를 탄 경우를 의미
  - 최선 경우 시간은 6 + 20 + 10 = 36분

등교 시간 분석

- 최악 경우
  - 열차에 승차하려는 순간, 열차의 문이 닫혀서 다음 열차를 기다려야 하고 다음 열차가 4분 후에 도착한다면, 최악 경우는 6 + 4 + 20 + 10 = 40분

등교 시간 분석

- 평균 경우
  - 대략 최악과 최선의 중간이라고 가정했을 때, 38분

등교 시간 분석

- 상각 분석
  - 단순히 1회의 등교 시간을 분석하는 것이 아니라, 예를 들어, 한 학기 동안의 등교 시간을 분석해본다는 점에서 그 의미를 가짐
  - 한 학기 동안 100번 학교에 갔는데 대부분은 지하철을 이용하지만 실제로 지하철보다 시간이 오래 걸리지만 버스를 타고 가끔 학교에 갈 때도 있다면
  - 최악 경우 분석은 버스를 타고 등교하는 시간이 60분이라면 60x100 = 6,000분을 넘지 않는 것으로 표현
  - 상각 분석은 한 학기 동안 학교에 가는데 소요된 시간을 모두 합해서 학교에 간 횟수인 100으로 나눈 값을 1회 등교 시간으로 분석
  - 대표적인 분석 예제: 크러스컬(Kruskal)의 최소 신장 트리 알고리즘

2.6 복잡도의 점근적 표기

- 시간 복잡도는 입력 크기에 대한 함수로 표기
  - 함수는 여러 개의 항을 가지는 다항식
  - 이를 입력의 크기에 대한 함수로 표현하기 위해 점근적 표기(Asymptotic Notation)를 사용

- 점근적 표기
  - 입력 크기 n이 무한대로 커질 때의 복잡도를 간단히 표현하기 위해 사용하는 표기법
  - O(Big-Oh)-표기
  - Ω(Big-Omega)-표기
  - Θ(Theta)-표기

O(Big-Oh)-표기

- O-표기의 정의
  - 모든 $n \geq n_0$에 대해서 $f(n) \leq cg(n)$이 성립하는 양의 상수 c와 n_0가 존재하면, $f(n) = O(g(n))$이다.

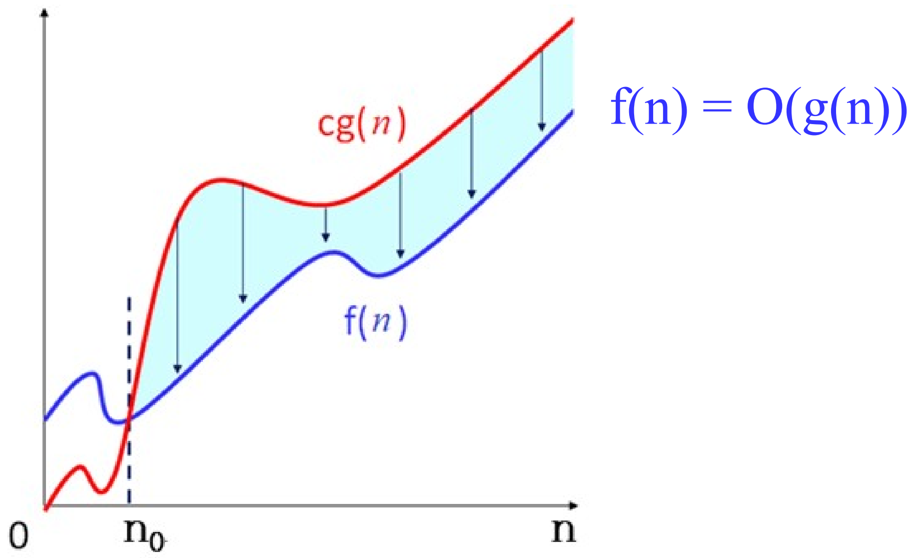
- O-표기의 의미
  - n_0와 같거나 큰 모든 n (즉, n_0 이후의 모든 n)에 대해서 f(n)이 cg(n)보다 크지 않다는 것

- $f(n) = O(g(n))$은 n_0 보다 큰 모든 n 대해서 f(n)이 양의 상수를 곱한 g(n)에 미치지 못한다는 뜻

- g(n)을 f(n)의 상한(Upper Bound)이라고 한다.

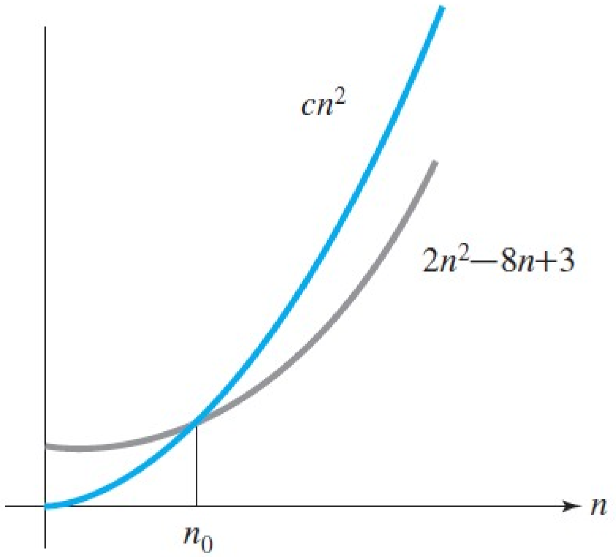
O(Big-Oh)-표기

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



O(Big-Oh)-표기

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



  - c=5일 때, $f(n) = 2n^2-8n+3$과 $5n^2$과의 교차점(n_0=1/3)이 생기는데, 이 교차점 이후 모든 n에 대해, 즉 n이 무한대로 증가할 때, $f(n) = 2n^2-8n+3$은 $5n^2$보다 절대로 커질 수 없다.
  - 따라서 $O(n^2)$이 $2n^2-8n+3$의 점근적 상한이 된다.

O-표기법 찾는 간단한 방법

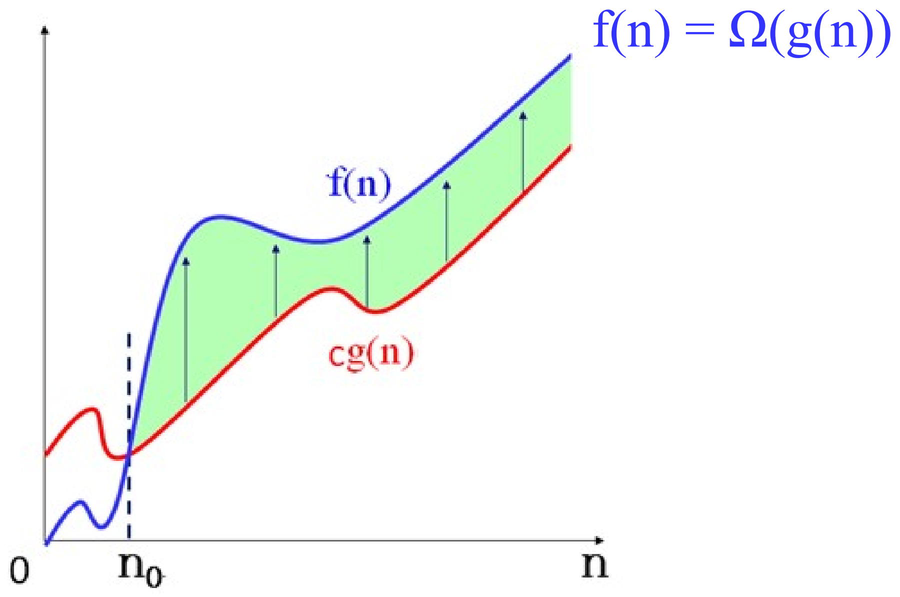
- 다항식에서 최고 차수 항만을 취한 뒤, 그 항의 계수를 제거하여 $g(n)$을 정한다.

Ω(Big-Omega)-표기

- Ω-표기의 정의
  - 모든 $n \geq n_0$에 대해서 $f(n) \geq cg(n)$이 성립하는 양의 상수 c와 n_0가 존재하면, $f(n) = \Omega(g(n))$이다.
- Ω-표기의 의미
  - n_0 보다 큰 모든 n 대해서 f(n)이 cg(n)보다 작지 않다는 것
- $f(n) = \Omega(g(n))$은 양의 상수를 곱한 g(n)이 f(n)에 미치지 못한다는 뜻
- g(n)을 f(n)의 하한(Lower Bound)이라고 한다.

Ω(Big-Omega)-표기

- $f(n) = \Omega(g(n))$
  - n이 증가함에 따라 $\Omega(g(n))$이 점근적 하한이라는 것 (즉, $g(n)$이 n_0보다 큰 모든 n에 대해서 항상 $f(n)$보다 작다는 것)을 보여준다.



Θ(Theta)-표기

- Θ-표기의 정의
  - 모든 $n \ge n_0$에 대해서 $c_1 g(n) \ge f(n) \ge c_2 g(n)$이 성립하는 양의 상수 c_1, c_2, n_0가 존재하면, $f(n) = \Theta(g(n))$이다.

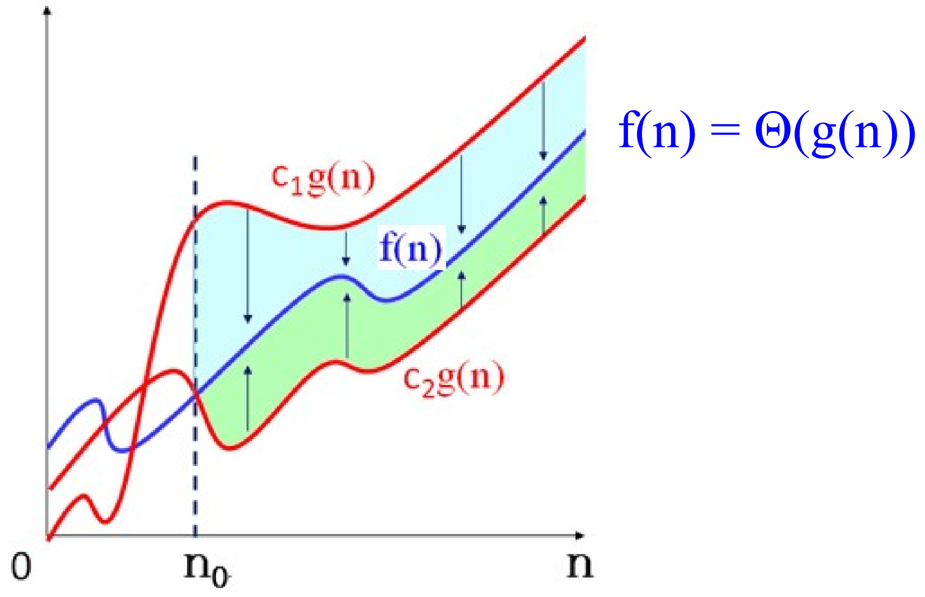
- Θ-표기의 의미
  - 수행시간의 O-표기와 Ω-표기가 동일한 경우에 사용한다. 즉 동일한 증가율을 의미

- $2n^2+3n+5=O(n^2)$과 동시에 $2n^2+3n+5=\Omega(n^2)$ 이므로, $2n^2+3n+5=\Theta(n^2)$

- $\Theta(n^2)$은 n^2과 $(2n^2+3n+5)$이 유사한 증가율을 가지고 있다는 뜻
  - 따라서 $2n^2+3n+5 \ne \Theta(n^3)$ 이고, $2n^2+3n+5 \ne \Theta(n)$이다.

Θ(Theta)-표기

- $f(n)=\Theta(g(n))$
  - n_0보다 큰 모든 n에 대해서 Θ-표기가 상한과 하한을 동시에 만족한다는 것을 보여준다.



O(Big-Oh)-표기 예

- $n \geq 2,\ 3n+2 \leq 4n \ \Rightarrow\ 3n+2=O(n)$
- $n \geq 3,\ 3n+3 \leq 4n \ \Rightarrow\ 3n+3=O(n)$
- $n \geq 10,\ 100n+6 \leq 101n \ \Rightarrow\ 100n+6=O(n)$
- $n \geq 5,\ 10n^2+4n+2 \leq 11n^2 \ \Rightarrow\ 10n^2+4n+2=O(n^2)$
- $n \geq 4,\ 6*2^n+n^2 \leq 7*2^n \ \Rightarrow\ 6*2^n+n^2=O(2^n)$
- $n \geq 2,\ 3n+3 \leq 3n^2 \ \Rightarrow\ 3n+3=O(n^2)$
- $n \geq 2,\ 10n^2+4n+2 \leq 10n^4 \ \Rightarrow\ 10n^2+4n+2=O(n^4)$

Ω(Big-Omega)-표기 예

- $n \ge 1,\ 3n+2 \ge 3n \Rightarrow 3n+2 = \Omega(n)$
- $n \ge 1,\ 3n+3 \ge 3n \Rightarrow 3n+3 = \Omega(n)$
- $n \ge 1,\ 100n+6 \ge 100n \Rightarrow 100n+6 = \Omega(n)$
- $n \ge 1,\ 10n^2+4n+2 \ge n^2 \Rightarrow 100n^2+4n+2 = \Omega(n^2)$
- $n \ge 1,\ 6*2^n+n^2 \ge 2^n \Rightarrow 6*2^n+n^2 = \Omega(2^n)$

Θ(Theta)-표기 예

- $n \ge 2,\ 3n \le 3n+2 \le 4n \Rightarrow 3n+2 = \Theta(n)$
  - $c_1=3,\ c_2=4,\ n_0=2$
- $3n+3 = \Theta(n)$
- $10n^2+4n+2 = \Theta(n^2)$
- $6*2^n+n^2 = \Theta(2^n)$

자주 사용하는 O-표기

- **O(1)** 상수 시간(Constant time)
- **O(logn)** 로그 시간(Logarithmic time)
- **O(n)** 선형 시간(Linear time)
- **O(nlogn)** 로그 선형 시간(Log-linear time)
- **$O(n^2)$** 이차 시간(Quadratic time)
- **$O(n^3)$** 3차 시간(Cubic time)
- **$O(n^k)$** 다항식 시간(Polynomial time), k는 상수
- **$O(2^n)$** 지수 시간 (Exponential time)

O-표기의 포함 관계

$O(2^n)$

$O(n^2)$

$O(\log n)$

$14, \ 1000$ — $O(1)$

$3\log n + 6$ — $O(n)$

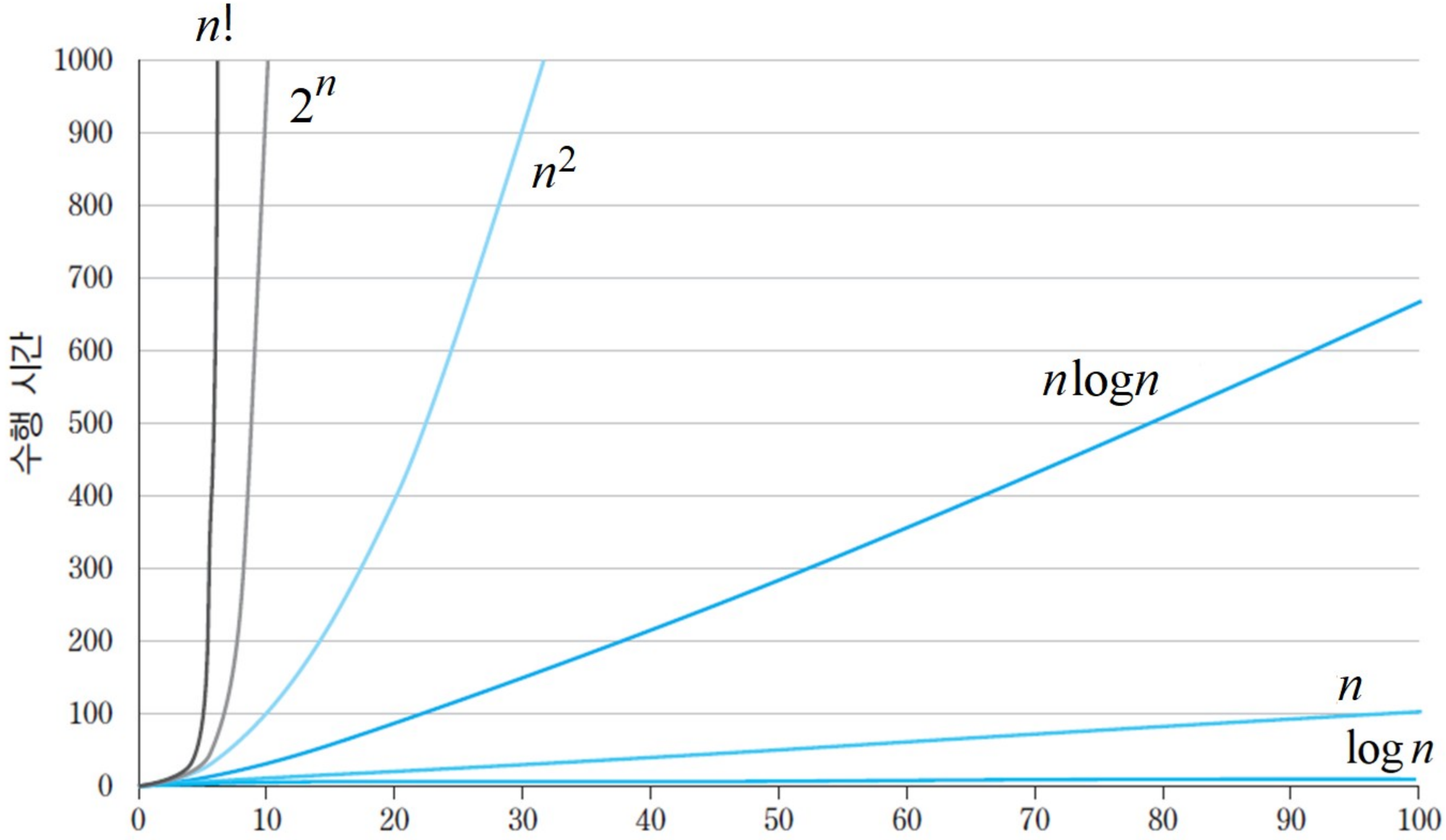
$5n+4$ — $O(n^3)$, $O(n^k)$

$2n^2+10n+3$

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

2^n

함수의 증가율 비교



n!
2^n
n^2
nlogn
n
log n
수행 시간
0
100
200
300
400
500
600
700
800
900
1000
0
10
20
30
40
50
60
70
80
90
100

2.7 효율적 알고리즘의 필요성

- 10억개를 정렬하는데 PC에서 O(n^2) 알고리즘은 300년, O(nlogn) 알고리즘은 5분

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

효율적 알고리즘의 필요성

- 효율적인 알고리즘은 슈퍼 컴퓨터보다 더 큰 가치가 있다.
- 값 비싼 H/W 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 주어야
  - 수행성: 각 단계는 컴퓨터에서 수행 가능하여야.
  - 유한성: 유한 시간 내에 종료되어야
  - 효율성: 효율적일수록 그 가치가 높다.

- 알고리즘은 대부분 의사 코드(pseudo code) 형태로 표현된다.

- 알고리즘의 효율성은 주로 시간 복잡도 (Time Complexity)가 사용된다.

- 시간 복잡도는 알고리즘이 수행하는 기본적인 연산 횟수를 입력 크기에 대한 함수로 표현

- 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