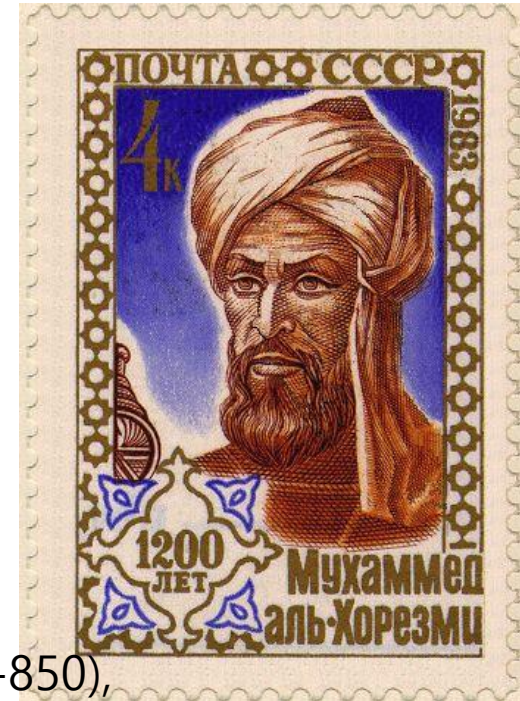


# 1장. 알고리즘:효율, 분석 그리고 차수

(Algorithm: efficiency, analysis, and order)

# 알고리즘 단어의 기원

Stamp issued by the Soviet Union on Sept. 6, 1983, to mark the 1200th birth anniversary of Al'Khowârizmî. (Image courtesy of Jeff Miller)



- Abu Ja'far Mohammed ibn Mûsâ al'Khowârizmî (780-850),
- "Father of Ja'far, Mohhamed, son of Moses, native of Khowârizm."
- 페르시아의 수학자, 천문학자, 지리학자
- Persian mathematician, astronomer and geographer during the Abbasid Caliphate, a scholar in the House of Wisdom in Baghdad.

Q1. 24와 16에 대해

- 최대공약수?
- 최소공배수?

- 우리는 각 문제를 해결하는 절차를 알고 있다.

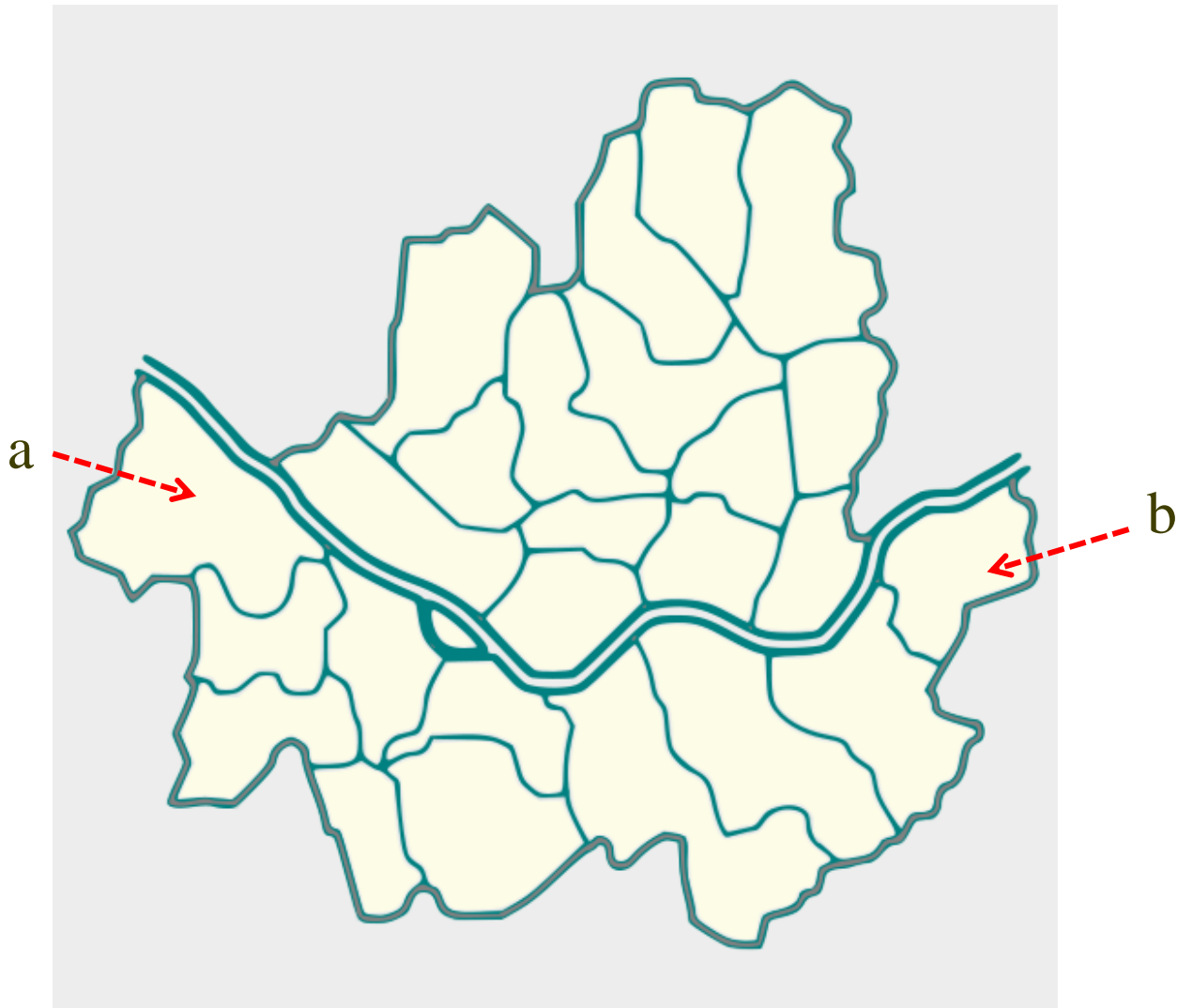
2		24	16
2		12	8
2		6	4
		3	2

최대공약수:  $2 \times 2 \times 2 = 8$

최소공배수:  $2 \times 2 \times 2 \times 3 \times 2 = 48$

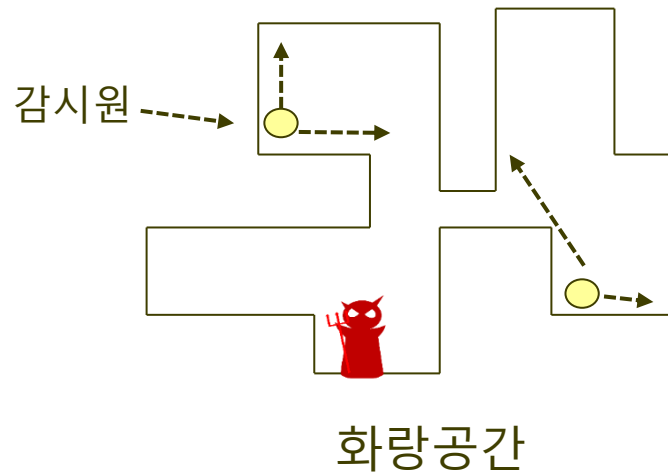
- 알고 있는 이 방법을 어떻게 설명할 수 있나?

Q2. a에서 b로 가는 최단거리?



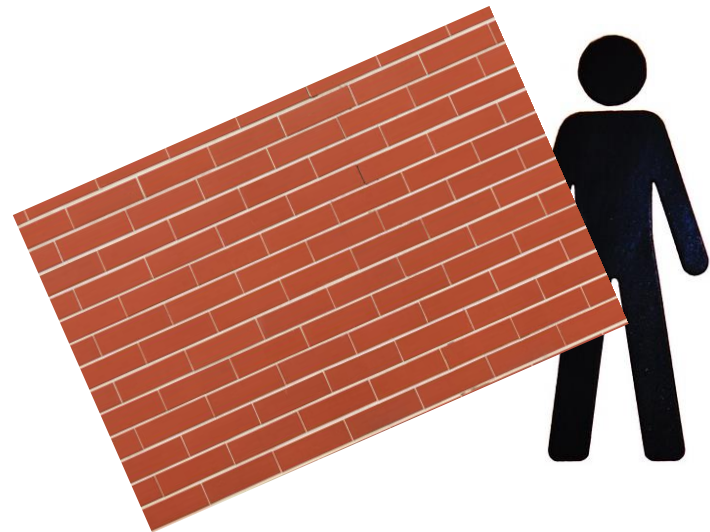
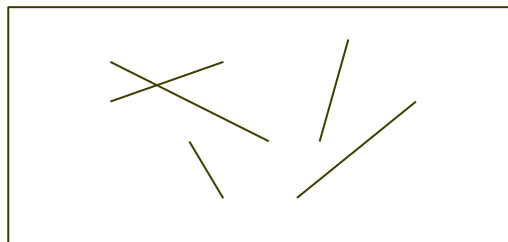
### Q3. 화랑 문제(Art Gallery Problem):

모든 공간을 감시하기 위한 최소 인원 배치는?



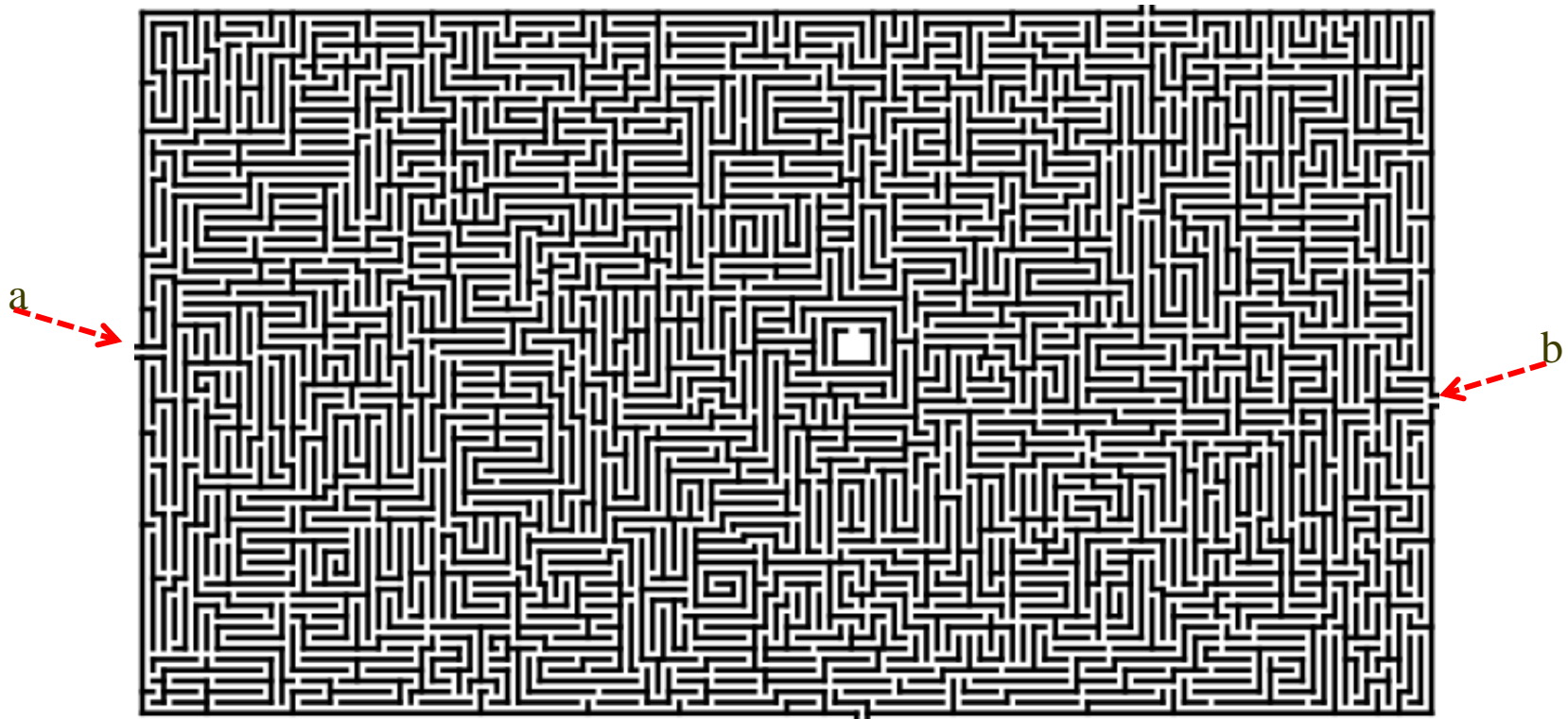
#### Q4. 선분교차문제

- 컴퓨터 그래픽에서 평면 또는 3차원 공간상의 두 선분이 교차하는지 확인하는 방법은?
- 평면과 선분이 어느 지점에서 교차하는지 확인하는 방법은?



벽과 사람의 충돌 장면 표현

Q5. a에서 b로 가는 길?



Mysid (SVG), Ilkant (original)

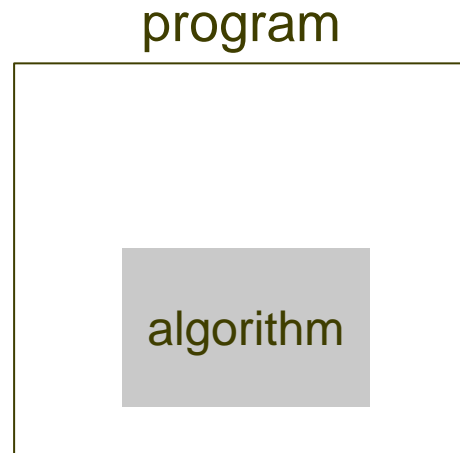
## 정의: 알고리즘(algorithm)

- 문제를 해결할 수 있는 잘 정의된(well defined) 유한 (finite)시간 내에 종료되는 계산적인(computational) 절차(procedure)
- 입력을 받아서 출력으로 변환시켜주는 일련의 계산절차



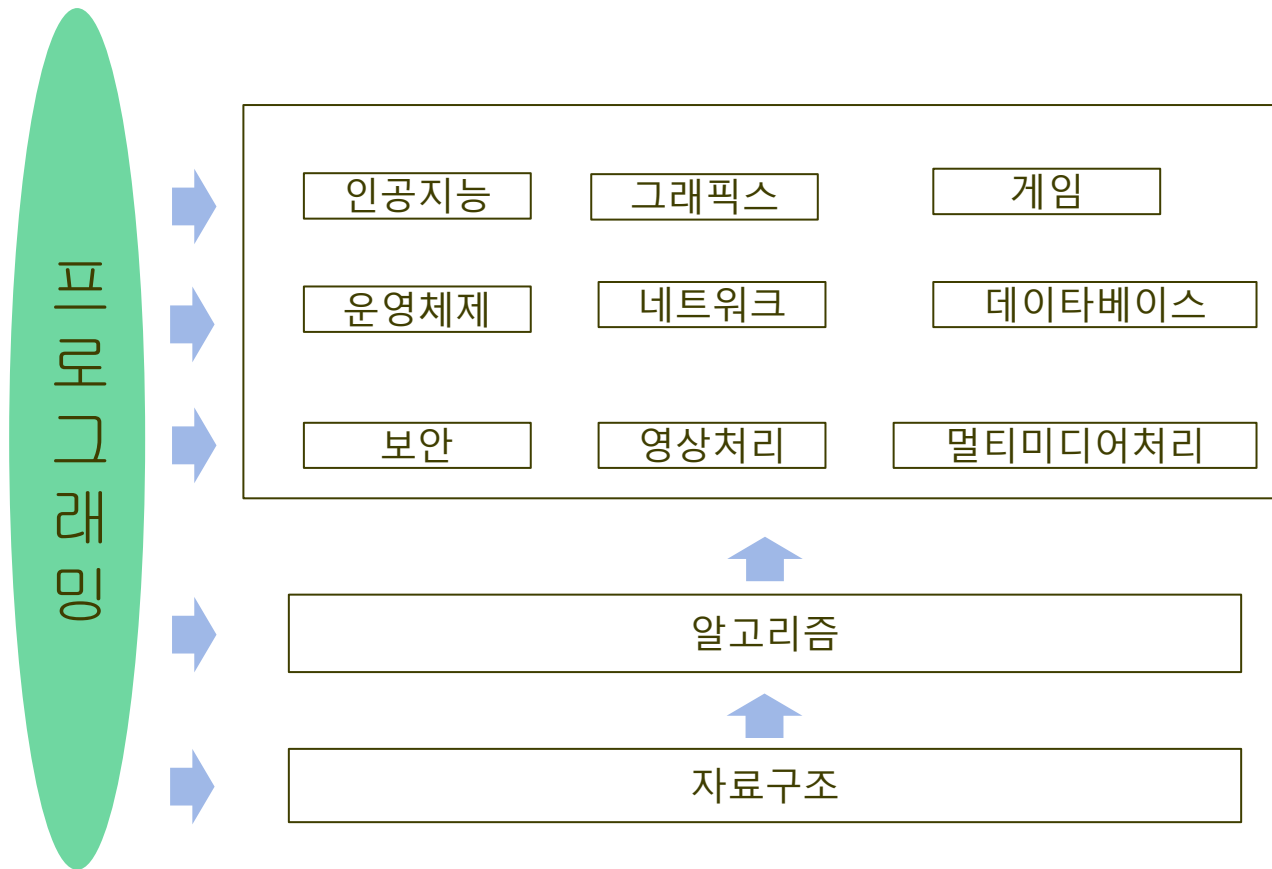
- 알고리즘은 계산과 데이터 처리에 사용되는 일련의 명령들이다.
- 문제를 해결할 수 있는 잘 정의된(well defined) 유한 (finite) 시간 내에 종료되는 계산적인(computational) 절차(procedure)
- 작업을 수행하기 위해 만들어진 잘 정의된 명령들이 주어진 초기 상태에서 출발하여 잘 정의된 이후의 상태들로 변화하면서 종료상태로 되는 효과적인 방법
- 하나의 상태에서 다음 상태로 바뀌는 것은 반드시 결정적이어야 하는 것은 아니다. 즉, 확률적(probabilistic) 알고리즘과 같은 알고리즘은 무작위성을 포함한다. (from *Wikipedia*)

- It is formally a type of **effective** method in which a list of **well-defined** instructions for completing a task will, when given an initial state, proceed through a well-defined series of successive states, eventually **terminating** in an end-state.
- The transition from one state to the next is not necessarily deterministic; some algorithms, known as probabilistic algorithms, incorporate randomness. (from *Wikipedia*)

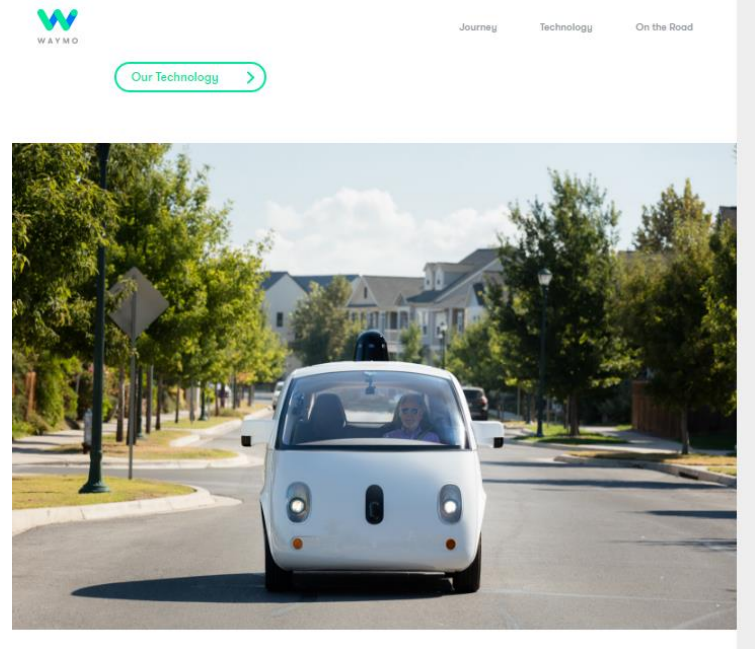


- 알고리즘은 프로그램의 엔진에 해당하는 중요한 절차이다.

## 왜 알고리즘분석 과목이 '전공필수'인가?



- 앞으로 기술의 발전은?
  - ✓ 4차산업혁명 - 새로운 IT 기술 출현  
AI, 빅데이터, IoT, 클라우드
  - ✓ 새로운 문제의 출현



waymo.com

google self-driving car



[Home](#)

[Research](#)

[Applied](#)

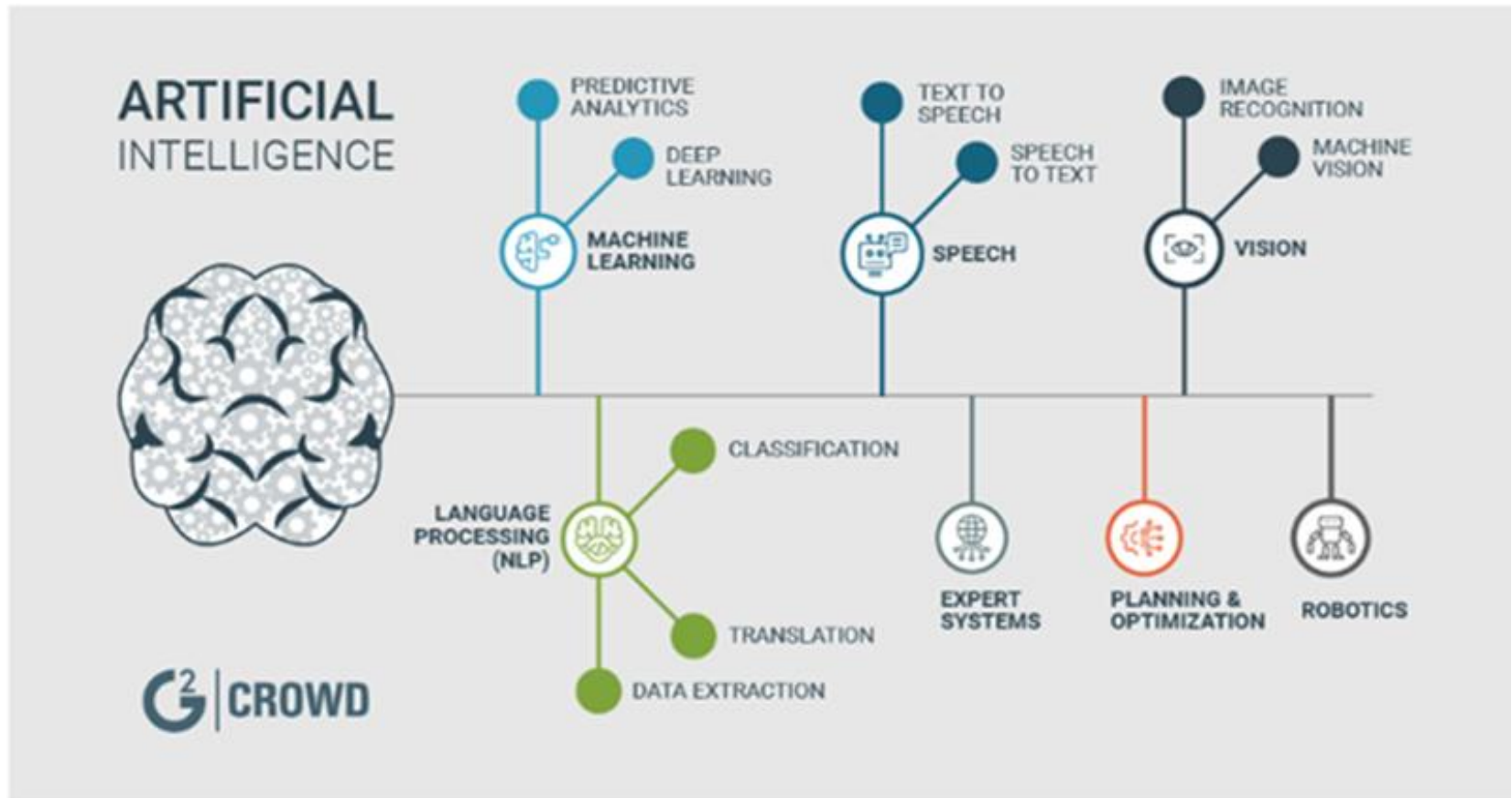
[News & Blog](#)

[About Us](#)

[Careers](#)



# AlphaGo



from <https://www.g2crowd.com/categories/artificial-intelligence>



## 메타버스(Metaverse)



[https://newsis.com/view/?id=NISX20220103\\_0001711562&cID=13001&pID=13000](https://newsis.com/view/?id=NISX20220103_0001711562&cID=13001&pID=13000)



<https://www.news1.kr/articles/?4542201>

## 암호화폐(Cryptocurrency)



<https://en.wikipedia.org/wiki/Cryptocurrency>

## NFT(Non-Fungible Token): 대체불가능한 토큰



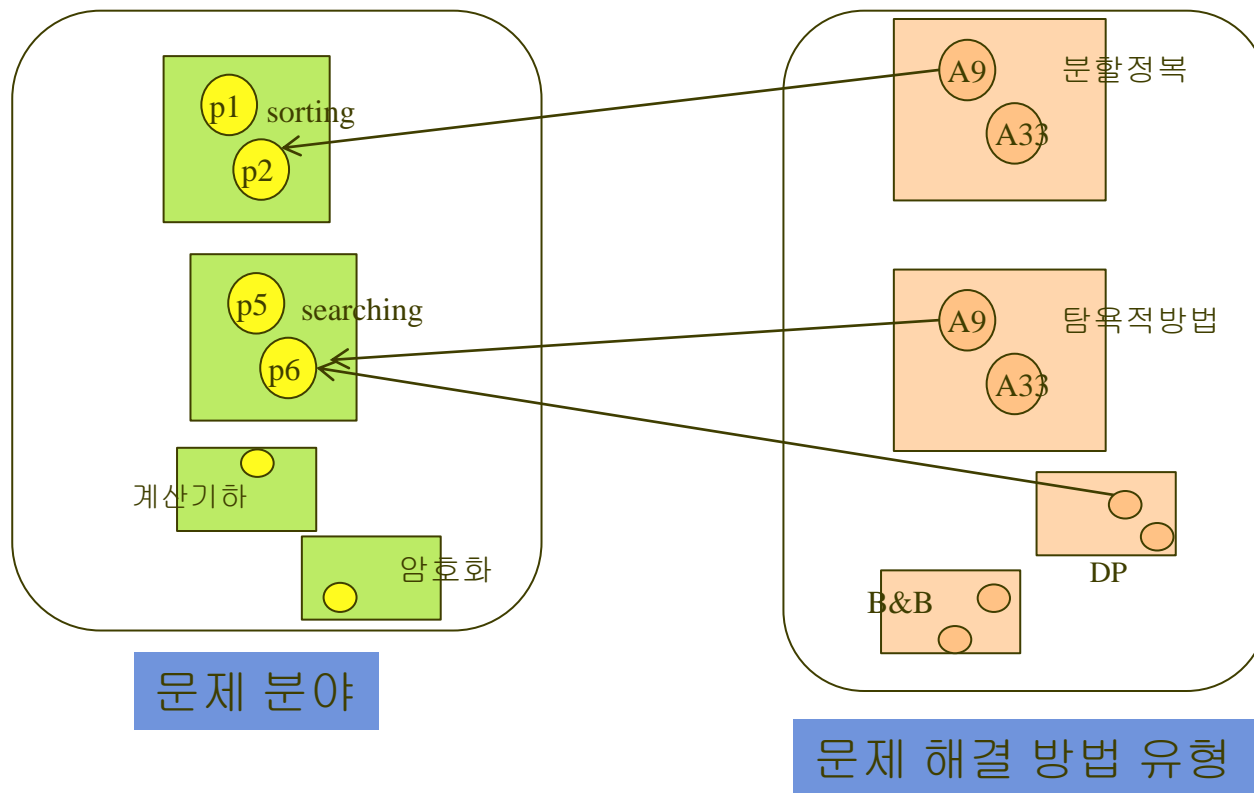
<https://commons.wikimedia.org/wiki/File:VeKings.png>

# 학습 목표

- Design(설계)
  - ✓ 알고리즘을 설계하는 기법을 배운다.
- Analysis(분석)
  - ✓ 알고리즘을 분석하여 시간/공간 복잡도를 구하는 방법을 배운다.
- Computational Complexity(계산복잡도)
  - ✓ 문제를 분석하여 계산적 복잡도를 구하는 방법을 배운다.
- Application Capability(응용력)
  - ✓ 새로운 문제를 분석하여 효과적인 알고리즘을 개발할 수 있는 능력을 배양

## 목 차

1. algorithm: efficiency, analysis, order(차수)
2. divide and conquer(분할 정복)
3. dynamic programming(동적계획법)
4. greedy algorithm(탐욕적인 방법)
5. backtracking(되추적)
6. branch and bound(분기한정법)
7. sorting(정렬)
8. searching(검색)
9. complexity and intractability(계산복잡도와 다루기 힘든 정도)

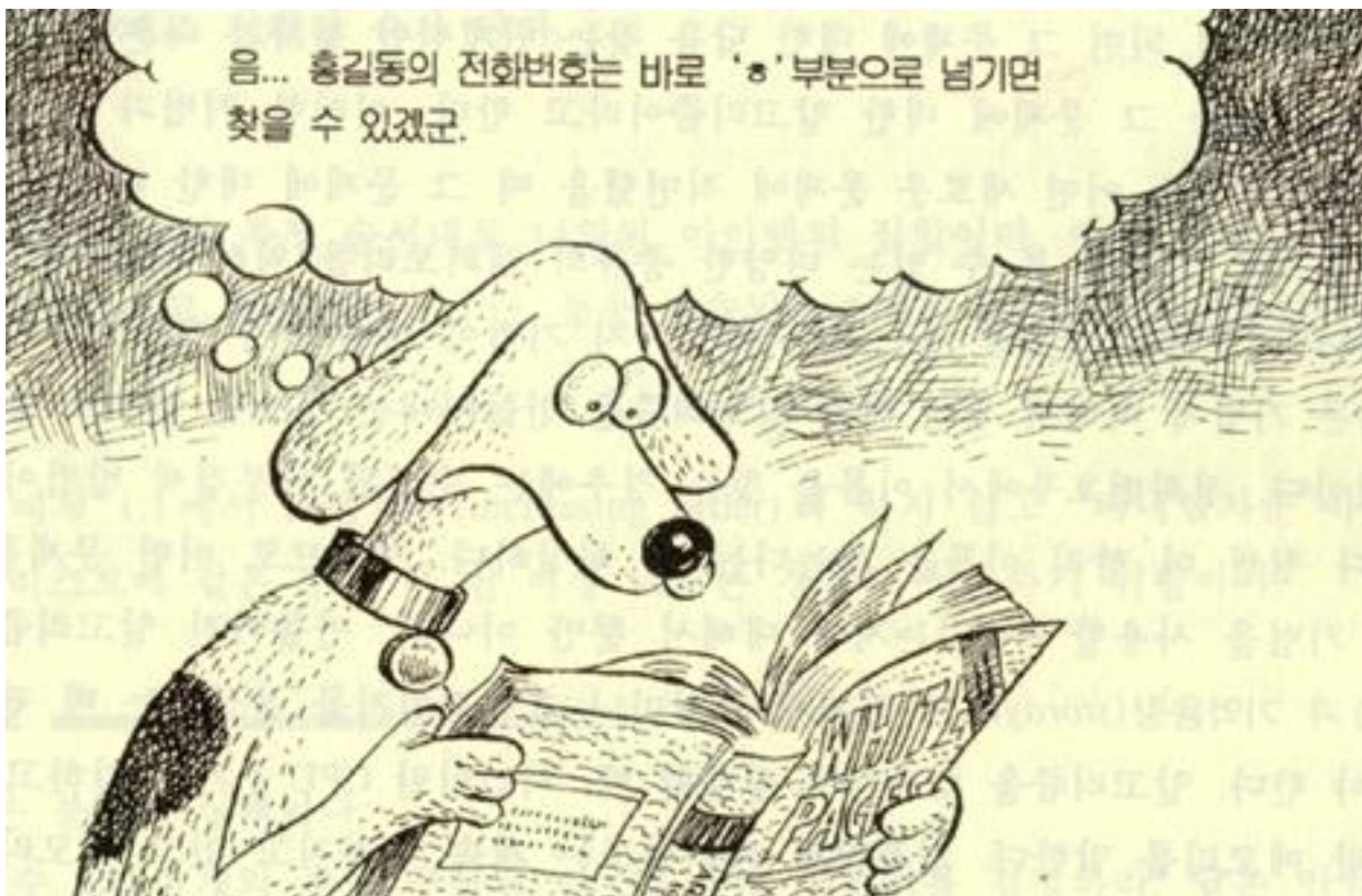


- ✓ 문제해결 방법 유형에 따라
  - divide and conquer, dynamic programming, greedy algorithm, backtracking, branch and bound, .....
- ✓ 문제의 분야에 따라
  - sorting, searching, graph problems, string algorithms, cryptography, data compression, game, computational geometric algorithms, optimization, ...

# 알고리즘과 Method의 차이

Algorithm	Method
유한시간 내에 종료	유한시간 내에 종료하는 지 모름

음... 홍길동의 전화번호는 바로 '8' 부분으로 넘기면  
찾을 수 있겠군.



# 알고리즘의 예

- 문제: 전화번호부에서 “홍길동”의 전화번호 찾기
- 알고리즘
  - ✓ 순차검색: 첫 쪽부터 홍길동이라는 이름이 나올 때까지 순서대로 찾는다.
  - ✓ 수정된 이분검색: 전화번호부는 “가나다”순으로 되어 있으므로 먼저 “ㅎ”이 있을 만한 곳으로 넘겨본 후 앞뒤로 뒤적여가며 찾는다. – 보간검색법(interpolation search)
- 분석: 어떤 알고리즘이 더 좋은가?



# 문제의 표기 방법

- 문제: 답을 찾고자 던지는 질문
- 파라미터(parameter): 문제에서 특정값이 주어지지 않은 변수 - 매개변수
- 문제의 사례(instance) (입력): 파라미터에 특정 값을 지정한 것
- 사례에 대한 해답(solution) (출력): 주어진 사례에 관한 질문에 대한 답

## 예제 1.1

# 문제의 예(정렬)

- 문제:  $n$ 개의 수로 구성된 리스트  $S$  를 비내림차순(nondecreasing order)으로 정렬(sort)하라.
- 파라미터:  $S, n$
- 사례(instance) – 파라미터에 어떤 값을 대입한 경우
  - 입력의 예:  $S = [10, 7, 11, 5, 13, 8], n = 6$
  - 출력의 예:  $S = [5, 7, 8, 10, 11, 13]$

## 예제 1.2

# 문제의 예(검색)

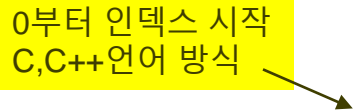
- 문제: 어떤 수  $x$ 가  $n$ 개의 수로 구성된 리스트  $S$ 에 있는지 결정하라.  $S$ 에 있으면 “예”, 없으면 “아니오”로 답하시오.
- 파라미터:  $S, n, x$
- 사례(instance)
  - 입력의 예:  $S = [10, 7, 11, 5, 13, 8], n = 6, x = 5$
  - 출력의 예: “예”

# 알고리즘의 표기

- 자연어: 한글 또는 영어 – 문제를 정확하게 기술하는데 어려움 있음. 상대적으로 긴 문장이 필요. 해석하는 사람에 따라 다르게 해석할 가능성 존재
- 프로그래밍언어: C, C++, Java, ML 등 – 너무나 구체적인 기술을 해야 하므로, 알고리즘을 이해하는데 어려움
- 의사코드(pseudo-code)
  - ✓ 의사(疑似): 실제와 비슷한
  - ✓ 직접 실행할 수 있는 프로그래밍언어는 아니지만, 거의 실제 프로그램에 가깝게 계산과정을 표현할 수 있는 언어
  - ✓ 간결하면서도 정확한 의미 표현 가능
- 알고리즘은 보통 의사코드로 표현한다.
- 이 강의에서는 C++에 가까운 의사코드를 사용한다.
  - ✓ 의미전달에 문제가 없을 경우에는 축약된 형태의 코드로 표시

[문제] 다음의 10개의 데이터에서 제일 큰 수를 찾는다.

0부터 인덱스 시작  
C,C++언어 방식



a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	3	8	2	9	4	7	1	0	6

- 배열(array)

공통의 성질을 갖는 변수가 여러 개 일 때 하나의 변수명을 정하고, 위치를 나타내는 인덱스를 이용해서 변수를 나타내는 자료구조 (data structure)

# 1. 자연어(natural language)

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	3	8	2	9	4	7	1	0	6

[배열에 저장된 10개의 수 중 최댓값 찾기]

배열의 제일 처음에 있는 데이터를 임시로 최댓값 M으로 설정한다. 오른쪽으로 이동하면서 보고 있는 데이터와 M과 비교해서, 보고 있는 데이터가 더 크면 그 데이터를 M으로 재설정하고, 아니면 다음의 오른쪽 데이터로 이동하면서 동일한 작업을 수행한다. 데이터의 끝까지 이동하면 M이 데이터의 최댓값이 된다.

## 2. 의사코드

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	3	8	2	9	4	7	1	0	6

(step 1) a[0]의 데이터를 임시로 최댓값 M으로 설정

(step 2) a[1]의 값과 M과 비교해서, a[1]이 더 크면 a[1]을 M  
으로 재설정, 아니면 a[2]값 비교로 이동

(step 3) step 2를 나머지 데이터에 대해서 수행

(step 4) M이 데이터의 최댓값

## 또 다른 의사코드

```
M = a[0]
for each a[i], 1 ≤ i ≤ 9
    if a[i] > M then
        M = a[i]
    fi
end for
M is maximum
```

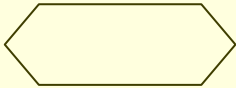


a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	3	8	2	9	4	7	1	0	6



### 3. 흐름도

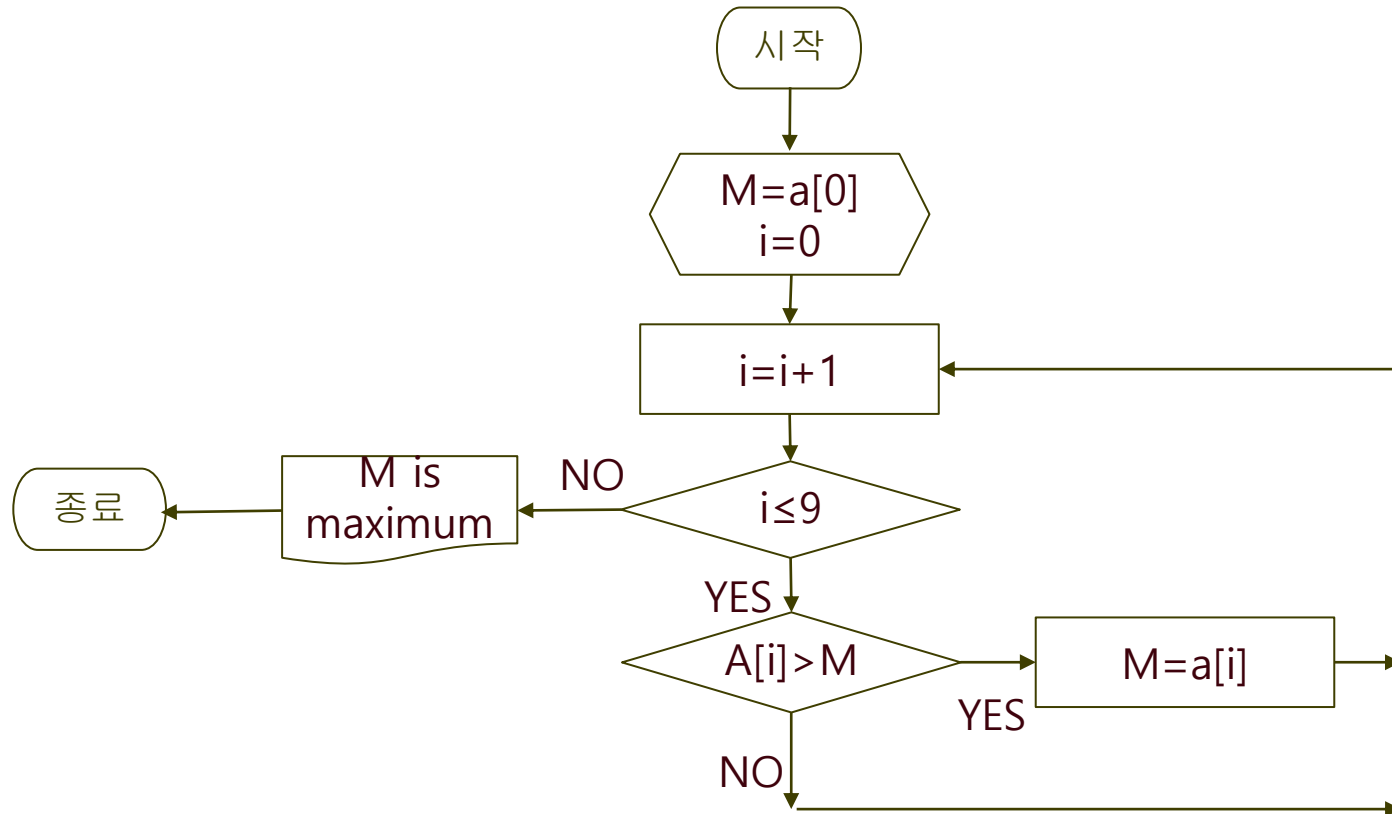
#### 흐름도에서 사용되는 표시

심벌 이름	도형	내용
터미널 심벌		흐름도의 시작과 끝을 나타내는 기호
입출력 심벌		입출력 작업 표시
프로세스 심벌		연산 명령문 등 처리해야 할 작업 내용
판단 심벌		판단을 나타낸다
연결심벌		페이지 내, 외의 연결을 나타낸다
미리 정의된 프로세스 심벌		모듈, 함수, 메소드, 하위절차
흐름심벌		연결 흐름 표시

심벌 이름	도형	내용
준비 심벌		변수의 초기화
문서출력 심벌		문서로 출력
데이터베이스 심벌		데이터베이스

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

5	3	8	2	9	4	7	1	0	6
---	---	---	---	---	---	---	---	---	---



## 4. 프로그래밍 언어(programming language)

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	3	8	2	9	4	7	1	0	6

C++ 예

```
void main() {  
    int i, a[10]={5,3,8,2,9,4,7,1,0,6};  
    int M;  
    M=a[0];  
    for(i=1;i<=9;i++)  
        if(a[i]>M) M=a[i];  
    cout << M;  
}
```

## Python 예

```
a=[5,3,8,2,9,4,7,1,0,6]
M=a[0]
for i in range(1,10):
    if(a[i]>M):
        M=a[i]
print(M)
```

# C++와 의사코드의 차이점(1)

- 배열 인덱스의 범위에 제한 없음
  - ✓ C++는 반드시 0부터 시작
  - ✓ 의사코드는 임의의 값 사용 가능
- 프로시저의 파라미터에 2차원 배열 크기의 가변성 허용
  - ✓ 예: `void pname(A[][]) { ... }`
- 지역배열에 변수인덱스 허용
  - ✓ 예: `keytype S[low..high];`

## C++와 의사코드의 차이점(2)

- 수학적 표현식 허용

- ✓ `low <= x && x <= high`  $\Rightarrow$  `low  $\leq$  x  $\leq$  high`

- ✓ `temp = x; x = y; y = temp`  $\Rightarrow$  `exchange x and y`

- C++에 없는 타입 사용 가능

- ✓ `index`: 첨자로 사용되는 정수 변수

- ✓ `number`: 정수(`int`) 또는 실수(`float`) 모두 사용가능

# C++와 의사코드의 차이점(3)

- 제어 구조

- ✓ `repeat (n times) { ... }`

- 프로시저와 함수

- ✓ 프로시저: `void pname(...) {...}`

- ✓ 함수: `returntype fname (...) {... return x;}`

- 참조파라미터(reference parameter)를 사용하여 프로시저의 결과값 전달

- ✓ 배열: 참조 파라미터로 전달

- ✓ 기타: 데이터타입 이름 뒤에 `&`를 붙임

- ✓ `const` 배열: 전달되는 배열의 값이 불변



# 순차검색 알고리즘 (sequential search)

- 문제: 크기가  $n$ 인 배열  $S$ 에  $x$ 가 있는가?
- 입력(파라미터): 양수  $n$ , 배열  $S[1..n]$ , 키  $x$
- 출력:  $x$ 가  $S$ 의 어디에 있는지의 위치. 만약 없으면 0.
- 알고리즘(자연어):
  - ✓  $x$ 와 같은 아이템을 찾을 때까지  $S$ 에 있는 모든 아이템을 차례로 검사한다.
  - ✓ 만일  $x$ 와 같은 아이템을 찾으면  $S$ 에서 위치를 리턴하고,  $S$ 를 모두 검사하고도 찾지 못하면 0을 리턴한다.

## Alg 1.1

## 순차검색 알고리즘

문제:  $n$ 개의 키로 구성된 배열  $S$ 에 키  $x$ 가 있는가?

입력: 양의 정수  $n$ , 1에서  $n$ 까지의 첨자를 가진 키의 배열  $S$ , 그리고  $x$

출력:  $S$ 안에  $x$ 의 위치를 가리키는 `location`( $S$ 안에  $x$ 가 없으면 0)

```
void seqsearch(int n,
               const keytype S[],
               keytype x,
               index& location) { // 출력
    location = 1;
    while (location <= n && S[location] != x)
        location++;
    if (location > n)
        location = 0;
}
```

✓ while-루프: 아직 검사할 항목이 있고,  $x$ 를 찾지 못했나?

✓ if-문: 모두 검사하였으나,  $x$ 를 찾지 못했나?

## Questions

- 순차검색 알고리즘으로 키를 찾기 위해서  $S$ 에 있는 항목을 몇 개나 검색해야 하는가?
  - ✓ 키와 같은 항목의 위치에 따라 다름
  - ✓ 최악의 경우:  $n$
- 좀 더 빨리 찾을 수는 없는가?
  - ✓  $S$ 에 있는 항목에 대한 정보가 없는 한 더 빨리 찾을 수 없다.

## Alg 1.2

## 배열의 수 더하기

문제:  $n$ 개의 수로 된 배열  $S$ 에 있는 모든 수를 더하라

입력: 양의 정수  $n$ , 수의 배열  $S$ (첨자는 1부터  $n$ )

출력:  $S$ 에 있는 수의 합,  $sum$

```
number sum(int n, const number S[ ]) {  
    index i;  
    number result;  
  
    result = 0;  
    for (i=1;i<=n;i++)  
        result = result+S[i];  
    return result;  
}
```

### Alg 1.3

## 교환정렬

문제: 비내림차순(nondecreasing order)으로  $n$ 개의 키를 정렬하라

입력: 양의 정수  $n$ , 키의 배열  $S$ (첨자는 1부터  $n$ )

출력: 키가 비내림차순으로 정렬된 배열  $S$

```
void exchangesort(int n, keytype S[ ]) {  
    index i, j;  
  
    for (i=1; i<=n-1; i++)  
        for (j=i+1; j<=n; j++)  
            if(S[j] < S[i])  
                exchange S[i] and S[j]  
}
```

## Alg 1.4

## 행렬곱셈

문제: 두 개의  $n \times n$  행렬의 곱을 구하라

입력: 양의 정수  $n$ , 수의 2차원 배열  $A$ 와  $B$ . 여기서 이 행렬의 행과 열은 모두 1부터  $n$ 까지의 첨자를 갖는다

출력:  $A$ 와  $B$ 의 곱이 되는 수의 2차원 배열  $C$ . 이 행렬의 행과 열은 모두 1부터  $n$ 까지의 첨자를 갖는다

```
void matrixmult(int n, const number A[ ][ ],
                const number B[ ][ ],
                number C[ ][ ]) {
    index i, j, k;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++) {
            C[i][j]=0;
            for (k=1; k<=n; k++)
                C[i][j]=C[i][j]+A[i][k]*B[k][j];
        }
}
```

# 이분검색 알고리즘 (binary search)

- 문제: 크기가  $n$ 인 정렬된 배열  $S$ 에  $x$ 가 있는가?
- 입력: 양수  $n$ , 배열  $S[1..n]$ , 키  $x$
- 출력:  $x$ 가  $S$ 의 어디에 있는지의 위치. 만약 없으면, 0.

## Alg 1.5

## 이분검색 알고리즘

```
void binsearch(int n,
               const keytype S[],
               keytype x,
               index& location) {    // 출력
    index low, high, mid;

    low = 1; high = n;
    location = 0;
    while (low <= high && location == 0) {
        mid = (low + high) / 2;    // 정수나눗셈
        if (x == S[mid])
            location = mid;
        else if (x < S[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
}
```

✓ while-루프: 아직 검사할 항목이 있고,  $x$ 를 찾지 못했나(location=0)?



## Questions

- 이분검색 알고리즘으로 키를 찾기 위해서  $S$ 에 있는 항목을 몇 개나 검색해야 하는가?
  - ✓ **while** 문을 수행할 때마다 검색 대상의 총 크기가 반 씩 감소하기 때문에 최악의 경우라도  $\lg n + 1$ 개만 검사하면 된다.

## 순차검색 vs. 이분검색 (\* 최악의 경우)

배열의 크기	순차검색의 비교 회수	이분검색의 비교회수
$n$	$n$	$\lg n + 1$
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

# $n$ 번째 피보나찌 수 구하기

피보나찌(Fibonacci) 수열의 정의

$$f_0 = 0$$

$$f_1 = 1$$

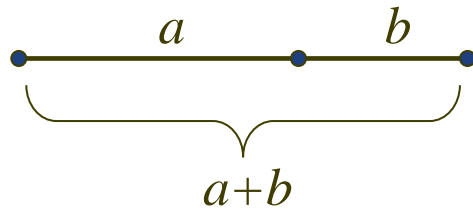
$$f_n = f_{n-1} + f_{n-2} \quad \text{for } n \geq 2$$

예: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, ...

$$f_n = \frac{[(1 + \sqrt{5})/2]^n - [(1 - \sqrt{5})/2]^n}{\sqrt{5}}$$

(예제 B.9 in appendix B)

## Golden Ratio



$$\frac{a+b}{a} = \frac{a}{b} = r$$

$$r = \frac{1 + \sqrt{5}}{2} = 1.61803.....$$

## Alg 1.6

# 피보나찌 수 구하기(재귀적 방법)

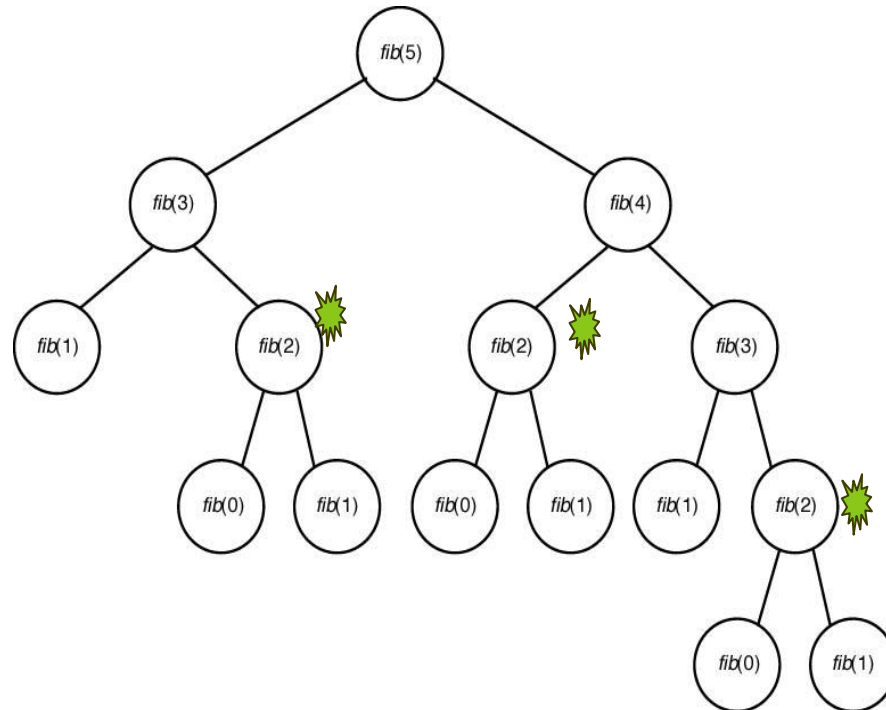
- 문제:  $n$ 번째 피보나찌 수를 구하라.
- 입력: 양수  $n$
- 출력:  $n$  번째 피보나찌 수
- 알고리즘:

```
int fib (int n) {  
    if (n <= 1)  
        return n;  
    else  
        return fib(n-1) + fib(n-2);  
}
```

## Discussion

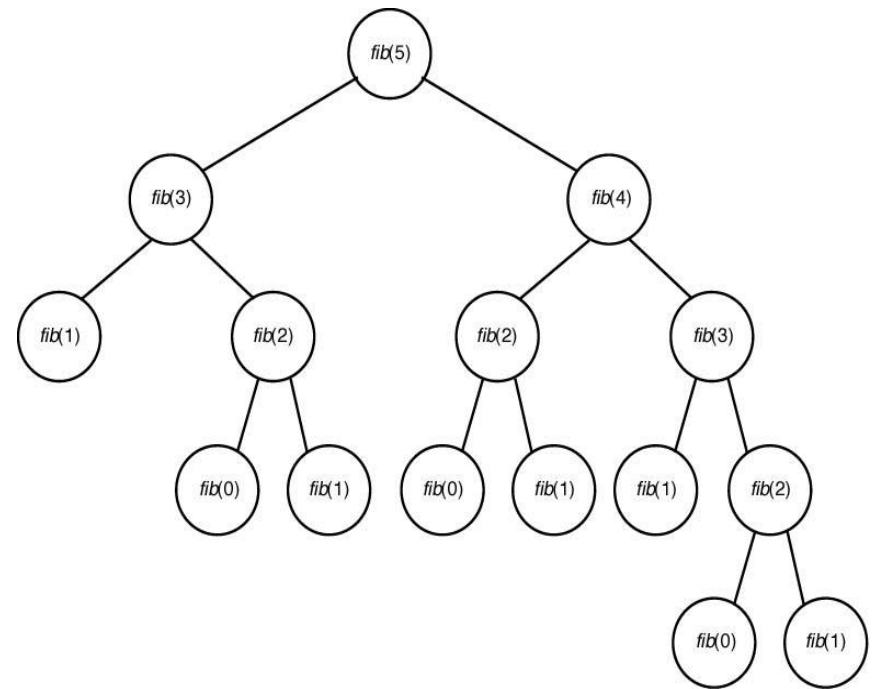
```
int fib (int n) {  
    if (n <= 1)  
        return n;  
    else  
        return fib(n-1) + fib(n-2);  
}
```

- 피보나찌 수 구하기 재귀 알고리즘은 수행속도가 매우 느리다.
  - ✓ 이유: 같은 피보나찌 수를 중복 계산
  - ✓ 예:  $\text{fib}(5)$  계산에  $\text{fib}(2)$  3번 중복계산



$\text{fib}(5)$ 의 재귀 트리

n	계산하는 항의 개수
0	1
1	1
2	3
3	5
4	9
5	15
6	25



$$25 = 15 + 9 + 1$$

```
int fib (int n) {
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);}
```

## fib(n)의 함수 호출 횟수 계산

$T(n) = \text{fib}(n)$ 을 계산하기 위하여 *fib* 함수를 호출하는 횟수  
즉, 재귀 트리 상의 마디의 개수

$$T(0) = 1$$

$$T(1) = 1$$

$$T(n) = T(n-1) + T(n-2) + 1 \quad \text{for } n \geq 2, n \text{이 짝수 가정}$$

$$> 2 \times T(n-2) \quad \text{왜냐하면 } T(n-1) > T(n-2)$$

$$> 2^2 \times T(n-4) \quad \text{왜냐하면 } T(n-2) > 2 \times T(n-4)$$

$$> 2^3 \times T(n-6)$$

...

$$> 2^{n/2} \times T(0)$$

$$= 2^{n/2}$$

자기 자신 포함

✓  $n$ 이 홀수일 때도 유사하게 증명 가능



# 수학적 귀납법 (mathematical induction)

1. 귀납출발점 (basis, base case):  $n = 0$ (또는 1) 일 때 주장이 사실임을 보임.
2. 귀납가정 (induction(inductive) hypothesis): 어떤  $n$ 에 대해서 주장이 사실임을 가정
3. 귀납절차(inductive step):  $n+1$ 에 대해서 주장이 사실임을 보임

$$S_n = \sum_{i=1}^n i = \frac{n(n+1)}{2} \quad \text{수학적 귀납법 증명}$$

1. 귀납출발점 (basis, base case):

$n = 1$  일 때

$$S_1 = 1 = \frac{1(2)}{2} \quad \text{성립}$$

2. 귀납가정 (induction(inductive) hypothesis):

$$k < n \text{ 인 모든 자연수 } k \text{에 대해 } S_k = \sum_{i=1}^k i = \frac{k(k+1)}{2} \quad \text{가정}$$

3. 귀납절차(inductive step):

$$\begin{aligned} S_n &= \sum_{i=1}^n i = S_{n-1} + n = \frac{(n-1)n}{2} + n \\ &= \frac{(n-1)n + 2n}{2} \\ &= \frac{n(n+1)}{2} \end{aligned}$$

$$* 1, 2, 3 \text{ 에 의해 모든 자연수 } n \text{에 대해 } S_n = \sum_{i=1}^n i = \frac{n(n+1)}{2} \quad \text{성립}$$

```
int fib (int n) {
    if (n <= 1)
        return n;
    else
        return fib(n-1) + fib(n-2);}
```

## 계산한 호출 횟수의 검증

**정리:** 재귀적 알고리즘으로 구성한 재귀 트리의 마디의 수를  $T(n)$  이라고 하면,  $n \geq 2$ 인 모든  $n$ 에 대하여  $T(n) > 2^{n/2}$  이다.

**증명:** ( $n$ 에 대한 수학적 귀납법으로 증명)

**귀납출발점:**  $T(2) = T(1) + T(0) + 1 = 3 > 2 = 2^{2/2}$

$T(3) = T(2) + T(1) + 1 = 5 > 2.83 \approx 2^{3/2}$

**귀납가정:**  $2 \leq m < n$ 인 모든  $m$ 에 대해서  $T(m) > 2^{m/2}$  이라 가정

**귀납절차:**  $T(n) > 2^{n/2}$ 임을 보이면 된다.

$$\begin{aligned}
 T(n) &= T(n-1) + T(n-2) + 1 \\
 &> 2^{(n-1)/2} + 2^{(n-2)/2} + 1 \\
 &> 2^{(n-2)/2} + 2^{(n-2)/2} \\
 &= 2 \times 2^{(n/2)-1} \\
 &= 2^{n/2}
 \end{aligned}$$

[귀납가정에 의하여]

## Alg 1.7

# 피보나찌 수 구하기 알고리즘 (반복적 방법)

```
int fib2 (int n) {  
    index i;  
    int f[0..n];  
    f[0] = 0;  
    if (n > 0) {  
        f[1] = 1;  
        for (i = 2; i <= n; i++)  
            f[i] = f[i-1] + f[i-2];  
    }  
    return f[n];  
}
```

## Discussion

- 반복 알고리즘은 수행속도가 훨씬 더 빠르다.
  - ✓ 이유: 중복 계산이 없음
- 계산하는 항의 총 개수
  - ✓  $T(n) = n + 1$
  - ✓ 즉,  $f[0]$ 부터  $f[n]$ 까지 한번씩 만 계산

## 두 피보나찌 알고리즘의 비교

$n$	$n+1$	$2^{n/2}$	반복(Alg 1.7)	재귀(Alg 1.6 하한)
40	41	1,048,576	41ns	1048 $\mu$ s
60	61	$1.1 \times 10^9$	61ns	1s
80	81	$1.1 \times 10^{12}$	81ns	18mins
100	101	$1.1 \times 10^{15}$	101ns	13days
120	121	$1.2 \times 10^{18}$	121ns	36years
160	161	$1.2 \times 10^{24}$	161ns	$3.8 \times 10^7$ years
200	201	$1.3 \times 10^{30}$	201ns	$4 \times 10^{13}$ years

Assume that 1 transaction takes 1 ns. (1 ns =  $10^{-9}$  second, 1  $\mu$ s =  $10^{-6}$  second)

✓ A사 휴대폰가입자수: 2,860만명

문제: 성명별 정렬

1 transaction => 1 ns

$n*n$ 알고리즘	$n*\lg n$ 알고리즘
817,960,000,000,000	$28,600,000*24 \approx 6,864,000,000$
817,960초=9.46 일	6.8초

✓ Big Data의 출현으로 많은 수의 자료 처리 필요

- 새로운 문제와 새로운 해법
- 효과적인 알고리즘의 중요성 증대

## Discussion

- Alg 1.6(재귀법)
  - ✓ 분할정복방법 (divide-and-conquer)
  - ✓ 2장에서 다룸
  - ✓ 어떤 문제에서는 매우 효율적
  - ✓ 피보나찌 문제에서는 비효율적
- Alg 1.7(반복법)
  - ✓ 동적계획법(dynamic programming)
  - ✓ 3장에서 다룸
- 문제에 따라 효율적인 방법이 다를 수 있음.



# 알고리즘의 분석(analysis)(1)

- 공간복잡도(space(memory) complexity) 분석
  - ✓ 입력크기에 따라서 작업공간 (메모리)이 얼마나 필요한 지 결정하는 절차
- 시간복잡도(time complexity) 분석
  - ✓ 입력크기에 따라서 단위연산이 몇 번 수행되는지 결정하는 절차
  - ✓ 알고리즘이 수행되는 기계에 따라 문제를 해결하는 시간이 달라짐.

우리의 관심도  
시간복잡도 > 공간복잡도

## 알고리즘의 분석(2)

- 시간복잡도분석의 기준
  - ✓ 기계에 독립적인, 문제 본연의 복잡도를 표현하여야 함.
- 표현 척도
  - ✓ 단위연산(basic operation)
    - ✓ 비교문(comparison), 지정문(assignment) 등
  - ✓ 입력크기(input size)
    - ✓ 배열의 크기, 리스트의 길이, 행렬에서 행과 열의 크기, 트리에서 마디와 이음선의 수

# 분석 방법의 종류 (1)

1. Every-case analysis
2. Worst-case analysis
3. Average-case analysis
4. Best-case analysis

- 모든 경우 분석(every-case analysis)
  - ✓ 입력크기에만 종속. 입력값과는 무관
  - ✓ 입력 값과는 무관하게 결과 값은 항상 일정
- 최악의 경우 분석(worst-case analysis)
  - ✓ 입력크기와 입력값 모두에 종속
  - ✓ 단위연산이 수행되는 횟수가 최대인 경우 선택

## 분석 방법의 종류 (2)

- 평균의 경우 분석(average-case analysis)
  - ✓ 입력크기에 종속
  - ✓ 모든 입력에 대해서 단위연산이 수행되는 기대치(평균)
  - ✓ 각 입력에 대해서 확률 할당 가능
  - ✓ 일반적으로 최악의 경우보다 계산이 복잡
- 최선의 경우 분석(best-case analysis)
  - ✓ 입력크기와 입력 값 모두에 종속
  - ✓ 단위연산이 수행되는 횟수가 최소인 경우 선택

## Discussion

- 우리의 주요 관심은
  - ✓ Worst-case analysis
  - ✓ 비관적 시각으로 알고리즘을 고안
- 평균시간 개념의 비현실성
  - ✓ 핵발전소 통제 알고리즘의 평균소요시간



조건: 위험 감지 시 통제 알고리즘이 안전을 위해 2초 이내 수행되어야 한다

- 99%는 1초 소요
- 1%는 10초 소요
- 평균 1.09초 소요
- 이 알고리즘을 핵발전소에서 사용할 수 있을까?

- ✓ 버스 환승 시스템

## Alg 1.2

## 배열의 수 더하기

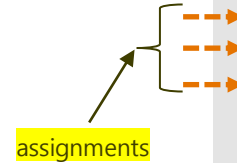
문제:  $n$ 개의 수로 된 배열  $S$ 에 있는 모든 수를 더하라

입력: 양의 정수  $n$ , 수의 배열  $S$ (첨자는 1부터  $n$ )

출력:  $S$ 에 있는 수의 합,  $sum$

```
number sum(int n, const number S[]) {  
    index i;  
    number result;  
  
    result = 0;  
    for (i=1;i<=n;i++)  
        result = result+S[i];  
    return result;  
}
```

# 배열 덧셈 시간 복잡도 분석



```
number sum(int n, const number S[]) {  
    index i;  
    number result;  
    result = 0;  
    for (i=1; i<=n; i++)  
        result = result+S[i];  
    return result;  
}
```

- 단위연산: 덧셈
- 입력크기: 배열의 크기  $n$
- 모든 경우 분석:
  - ✓ 배열 내용에 상관없이 for-루프가  $n$ 번 반복된다.
  - ✓ 각 루프마다 덧셈이 1회 수행된다.
  - ✓ 따라서,  $n$ 에 대해서 덧셈이 수행되는 총 횟수는  $T(n) = n$  이다.

- 단위연산: 지정문 (for-루프의 첨자 지정문 포함)
- 입력크기: 배열의 크기  $n$
- 모든 경우 분석:
  - ✓ 배열 내용에 상관없이 for-루프가  $n$ 번 반복된다.
  - ✓ 따라서, 지정문이  $T(n) = n + n + 1$ 번 수행된다.

### Alg 1.3

## 교환정렬

문제: 비내림차순(nondecreasing order)으로  $n$ 개의 키를 정렬하라

입력: 양의 정수  $n$ , 키의 배열  $S$ (첨자는 1부터  $n$ )

출력: 키가 비내림차순으로 정렬된 배열  $S$

```
void exchangesort(int n, keytype S[ ]) {  
    index i, j;  
  
    for (i=1; i<=n-1; i++)  
        for (j=i+1; j<=n; j++)  
            if(S[j] < S[i])  
                exchange S[i] and S[j]  
}
```



# 교환정렬 시간복잡도 분석(1)

- 단위연산: 조건문 ( $s[j]$  와  $s[i]$  의 비교)
- 입력크기: 정렬할 항목의 수  $n$
- 모든 경우 분석:
  - ✓  $j$ -루프가 수행될 때마다 조건문 1번씩 수행
  - ✓ 조건문의 총 수행횟수

```
void exchangesort(int n, keytype S[]){
    index i, j;
    for (i=1; i<=n-1; i++)
        for (j=i+1; j<=n; j++)
            if(S[j] < S[i])
                exchange S[i] and S[j]
}
```

$i$	$j$ loop 수행 회수
1	$n-1$
2	$n-2$
3	$n-3$
...	...
$n-1$	1

따라서 
$$T(n) = (n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2}$$

## 교환정렬 시간복잡도 분석(2)

- 단위연산: 교환하는 연산 (exchange S[j] and S[i])
- 입력크기: 정렬할 항목의 수  $n$
- 최악의 경우 분석:
  - ✓ 조건문의 결과에 따라서 교환 연산의 수행여부가 결정된다.
  - ✓ 최악의 경우 = 조건문이 항상 참(true)이 되는 경우  
= 입력 배열이 거꾸로 정렬되어 있는 경우

$$W(n) = \frac{(n-1)n}{2}$$

(예) 5, 4, 3, 2, 1

```
for (i=1; i<=n-1; i++)  
    for (j=i+1; j<=n; j++)  
        if (S[j] < S[i])  
            exchange S[i] and S[j]
```

# 행렬곱셈 시간복잡도 분석

- 단위연산: 가장 안쪽 for 루프에 있는 곱셈
- 입력크기: 행과 열의 개수  $n$
- 모든 경우 분석:
  - ✓ for  $i$  루프는 항상  $n$ 번 수행. for  $i$  루프 한 번 수행될 때마다 for  $j$  루프는 항상  $n$ 번 수행
  - ✓ for  $j$  루프 한 번 수행될 때마다 for  $k$  루프는 항상  $n$ 번 수행

$$T(n) = n \times n \times n = n^3$$

```
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j++) {  
        C[i][j]=0;  
        for (k=1; k<=n; k++)  
            C[i][j]=C[i][j]+A[i][k]*B[k][j];  
    }
```

# 순차검색 시간복잡도 분석 (최악)

- 단위연산: 배열의 아이템과 키  $x$ 와 비교 연산  
( $S[\text{location}] \neq x$ )
- 입력크기: 배열 안에 있는 아이템의 수  $n$
- 최악의 경우 분석:
  - ✓  $x$ 가 배열의 마지막 아이템이거나,  $x$ 가 배열에 없는 경우 단위 연산이  $n$ 번 수행된다.
  - ✓ 따라서,  $W(n) = n$

순차검색 알고리즘의 경우 입력배열의 값에 따라서  
검색하는 횟수가 달라지므로, 모든 경우 분석은 불가능하다.

```
location = 1;
while (location <= n && S[location] != x)
    location++;
if (location > n)
    location = 0;
```

# 순차검색 시간복잡도 분석 (평균)

- 단위연산: 배열의 아이템과 키  $x$ 와 비교 연산  
( $S[\text{location}] \neq x$ )
- 입력크기: 배열 안에 있는 아이템의 수  $n$
- 평균의 경우 분석:
  - ✓ 배열의 아이템이 모두 다르다고 가정한다.
  - ✓ 경우1 :  $x$ 가 배열  $S$ 안에 있는 경우만 고려
    - $1 \leq k \leq n$  에 대해서  $x$ 가 배열의  $k$ 번째 있을 확률  $= \frac{1}{n}$
    - $x$ 가 배열의  $k$ 번째 있다면, 이를 찾기 위해서 수행하는 단위 연산의 횟수  $= k$
    - 따라서, 
$$A(n) = \sum_{k=1}^n (k \times \frac{1}{n}) = \frac{1}{n} \times \sum_{k=1}^n k = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

```
location = 1;
while (location <= n && S[location] != x)
    location++;
if (location > n)
    location = 0;
```

✓ **경우2** :  $x$ 가 배열  $S$ 안에 없는 경우도 고려

- $x$ 가 배열  $S$ 안에 있을 확률을  $p$ 라고 하면,
  - $x$ 가 배열의  $k$ 번째 있을 확률  $= p/n$
  - $x$ 가 배열에 없을 확률  $= 1-p$

■ 따라서,

$$\begin{aligned} A(n) &= \sum_{k=1}^n \left( k \times \frac{p}{n} \right) + n(1-p) \\ &= \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p) \\ &= n\left(1 - \frac{p}{2}\right) + \frac{p}{2} \end{aligned}$$

있는 경우      없는 경우

- $p = 1 \Rightarrow A(n) = (n+1)/2$   
 $p = 1/2 \Rightarrow A(n) = 3n/4 + 1/4$

# 순차검색 시간복잡도 분석 (최선)

- 단위연산: 배열의 아이템과 키  $x$ 와 비교 연산  
(`S[location] != x`)
- 입력크기: 배열 안에 있는 아이템의 수  $n$
- 최선의 경우 분석:
  - ✓  $x$ 가  $S[1]$ 일 때, 입력의 크기에 상관없이 단위연산이 1번만 수행된다.
  - ✓ 따라서,

$$B(n) = 1$$

## Discussion

- 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석이 가장 정확한가?
- 최악, 평균, 최선의 경우 분석 방법 중에서 어떤 분석을 사용할 것인가?
- 복잡도 함수(complexity function)는 음이 아닌 정수가 주어지면 음이 아닌 실수를 내주는 함수
- $f(n) = n, \lg n, 3n^2 + 4n$  etc.



# 정확도(correctness) 분석

- 알고리즘이 의도한 대로 수행되는지를 증명하는 절차
- 정확도를 증명하는 것은 쉽지 않음
- 정확한 알고리즘이란?
  - ✓ 어떠한 입력에 대해서도 답을 출력하면서 멈추는 알고리즘
- 정확하지 않은 알고리즘이란?
  - ✓ 어떤 입력에 대해서 멈추지 않거나, 또는
  - ✓ 틀린 답을 출력하면서 멈추는 알고리즘

차수(order)

# 대표적인 복잡도 함수

- $\Theta(\lg n)$
- $\Theta(n)$  : 1차(linear)
- $\Theta(n \lg n)$
- $\Theta(n^2)$  : 2차(quadratic)
- $\Theta(n^3)$  : 3차(cubic)
- $\Theta(2^n)$  : 지수(exponential)
- $\Theta(n!)$
- $\Theta(n^n)$

2차 항이 궁극적으로 지배한다.

$n$	$0.1n^2$	$0.1n^2+n+100$
10	10	120
20	40	160
50	250	400
100	1,000	1,200
1,000	100,000	101,100

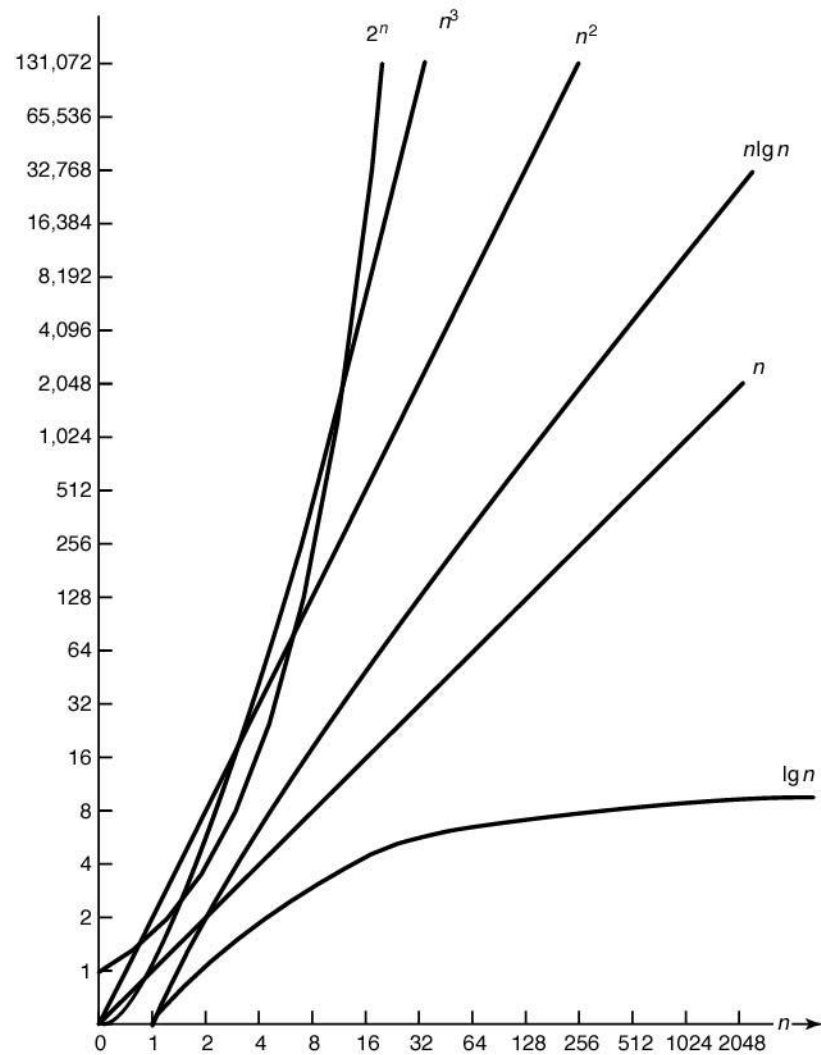
- Eventually

높은 차수항이 궁극적으로 지배한다.

$n$	$0.01n^2$	$100n$
10	1	1,000
100	100	10,000
1,000	10,000	100,000
10,000	1,000,000	1,000,000
100,000	100,000,000	10,000,000

10,000보다 큰  $n$ 에 대해서  $0.01n^2 > 100n$ .

# 복잡도 함수의 증가율



# 시간복잡도별 실행시간 비교

$n$	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 $\mu s^*$	0.01 $\mu s$	0.033 $\mu s$	0.1 $\mu s$	1 $\mu s$	1 $\mu s$
20	0.004 $\mu s$	0.02 $\mu s$	0.086 $\mu s$	0.4 $\mu s$	8 $\mu s$	1 ms <sup>†</sup>
30	0.005 $\mu s$	0.03 $\mu s$	0.147 $\mu s$	0.9 $\mu s$	27 $\mu s$	1 s
40	0.005 $\mu s$	0.04 $\mu s$	0.213 $\mu s$	1.6 $\mu s$	64 $\mu s$	18.3 min
50	0.006 $\mu s$	0.05 $\mu s$	0.282 $\mu s$	2.5 $\mu s$	125 $\mu s$	13 days
$10^2$	0.007 $\mu s$	0.10 $\mu s$	0.664 $\mu s$	10 $\mu s$	1 ms	$4 \times 10^{13}$ years
$10^3$	0.010 $\mu s$	1.00 $\mu s$	9.966 $\mu s$	1 ms	1 s	
$10^4$	0.013 $\mu s$	10 $\mu s$	130 $\mu s$	100 ms	16.7 min	
$10^5$	0.017 $\mu s$	0.10 ms	1.67 ms	10 s	11.6 days	
$10^6$	0.020 $\mu s$	1 ms	19.93 ms	16.7 min	31.7 days	
$10^7$	0.023 $\mu s$	0.01 s	0.23 s	1.16 days	31,709 years	
$10^8$	0.027 $\mu s$	0.10 s	2.66 s	115.7 days	$3.17 \times 10^7$ years	
$10^9$	0.030 $\mu s$	1 s	29.90 s	31.7 days		

\* 1  $\mu s = 10^{-6}$  second

† 1 ms =  $10^{-3}$  second

1초에  $10^9$  개 작업 처리

# Asymptotic(점근적) Behavior

- $f(n)$ 의 asymptotic behavior는  $n$ 이 큰 수가 될 때의 함수  $f(n)$ 이 갖는 특성
- (예)  $f(n) = 1/n$

$$\lim_{n \rightarrow \infty} \frac{1}{n} = 0$$



# 복잡도 함수 표기법

- $O()$  - big oh: asymptotic upper bound
- $o()$  - small oh: upper bound that is not asymptotically tight
- $\Omega()$  - omega: asymptotic lower bound
- $\omega()$  - small omega: lower bound that is not asymptotically tight
- $\Theta()$  - theta: asymptotic tight bound

# 큰(big) O 표기법

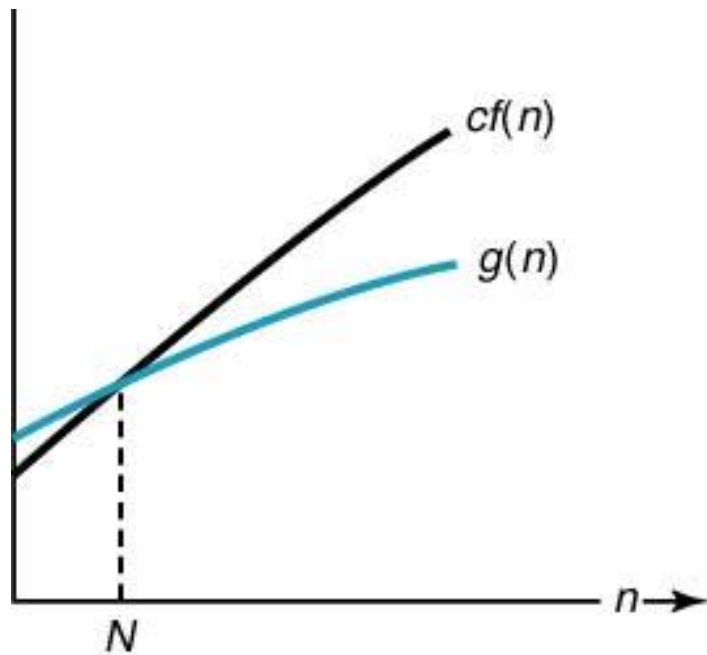
- 정의 : 점근적 상한(asymptotic upper bound)

- ✓ 주어진 복잡도 함수  $f(n)$ 에 대해서  $g(n) \in O(f(n))$  이면  
 $n \geq N$ 인 모든 정수  $n$ 에 대해서  $0 \leq g(n) \leq c \times f(n)$ 이 성립하는 양의 실수  $c$ 와 음이 아닌 정수  $N$ 이 존재한다.
- ✓  $O(f(n)) = \{g(n): \text{there exist positive constants } c \text{ and } N \text{ such that } 0 \leq g(n) \leq c \times f(n) \text{ for all } n \geq N\}$

- $g(n) \in O(f(n))$  읽는 방법:

- ✓  $g(n)$ 은  $f(n)$ 의 큰 오(big O)

## 큰 $O$ 표기법



(a)  $g(n) \in O(f(n))$

# 큰 $O$ 표기법

- 어떤 함수  $g(n)$ 이  $O(n^2)$ 에 속한다는 말은
  - ✓ 그 함수는  $n$ 이 커짐에 따라 (즉, 어떤 임의의  $N$ 값보다 큰 값에 대해서는) 어떤 2차 함수  $cn^2$  보다는 **작은** 값을 가지게 된다는 것을 뜻한다. (그래프 상에서는 **아래**에 위치)
  - ✓ 그 함수  $g(n)$ 은 어떤 2차 함수  $cn^2$  보다는 궁극적으로 **좋다**고 (빠르다) 말할 수 있다.
- 어떤 알고리즘의 시간복잡도가  $O(f(n))$ 이라면
  - ✓ 입력의 크기  $n$ 에 대해서 이 알고리즘의 수행시간은 **아무리 늦어도**  $cf(n)$ 은 된다. ( $cf(n)$ 이 점근적상한이다.)
  - ✓ 이 알고리즘의 수행시간은  $cf(n)$ 보다 절대로 더 느릴 수는 없다.

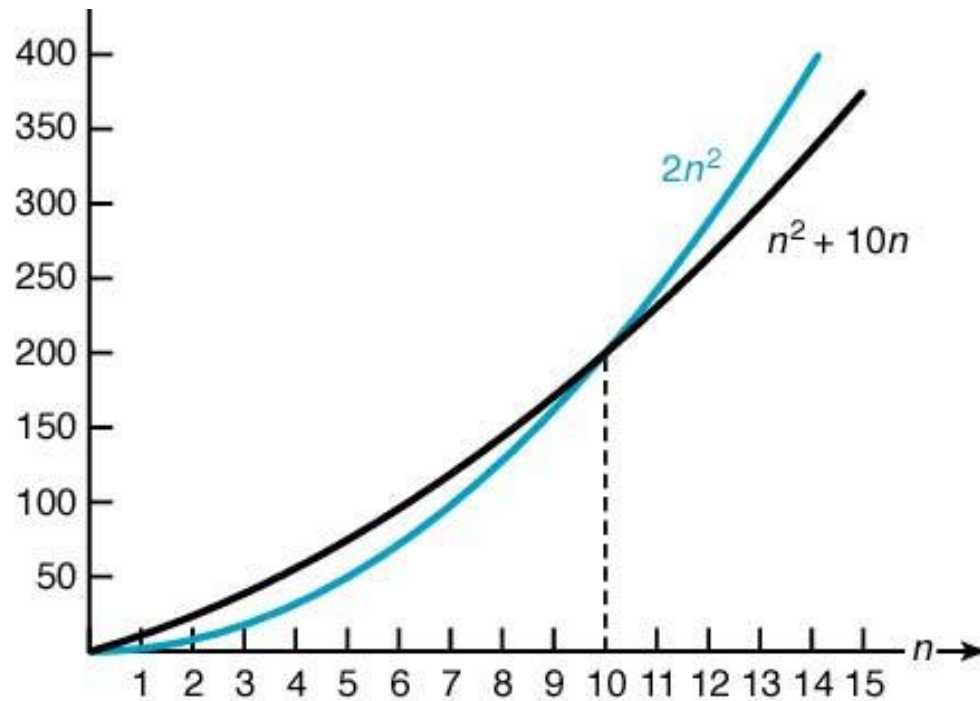
## 큰 $O$ 표기법 : 예

●  $n^2+10n \in O(n^2)$  ?

(1)  $n \geq 10$ 인 모든 정수  $n$ 에 대해서  $n^2+10n \leq 2n^2$  이 성립. 그러므로,  $c = 2$ 와  $N = 10$ 을 선택하면, “큰  $O$ ”의 정의에 의해서  $n^2+10n \in O(n^2)$ .

(2)  $n \geq 1$ 인 모든 정수  $n$ 에 대해서  $n^2+10n \leq n^2+10n^2 = 11n^2$  이 성립. 그러므로,  $c = 11$ 와  $N = 1$ 을 선택하면, “큰  $O$ ”의 정의에 의해서  $n^2+10n \in O(n^2)$ .

## $2n^2$ 과 $n^2 + 10n$ 의 비교



## 큰 $O$ 표기법 : 예 (계속)

- $5n^2 \in O(n^2)$  ?

$c=5$ 와  $N=0$ 을 선택하면,  $n \geq 0$ 인 모든 정수  $n$ 에 대해서  $5n^2 \leq 5n^2$  성립.

- $T(n) = \frac{n(n-1)}{2}$  ?

$n \geq 0$ 인 모든 정수  $n$ 에 대해서  $\frac{n(n-1)}{2} \leq \frac{n^2}{2}$  이 성립. 그러므로,  
 $c = \frac{1}{2}$  와  $N=0$ 을 선택하면,  $T(n) \in O(n^2)$ .

- $n^2 \in O(n^2+10n)$  ?

$n \geq 0$ 인 모든 정수  $n$ 에 대해서,  $n^2 \leq 1 \times (n^2+10n)$ 이 성립. 그러므로,  $c=1$ 와  $N=0$ 을 선택하면,  $n^2 \in O(n^2+10n)$ .

## 큰 $O$ 표기법 : 예 (계속)

- $n \in O(n^2)$  ?

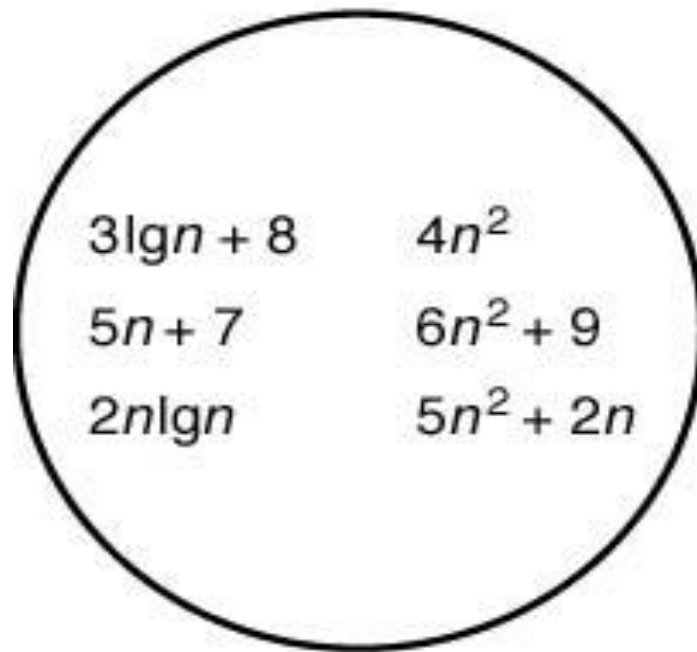
$n \geq 1$ 인 모든 정수  $n$ 에 대해서,  $n \leq 1 \times n^2$  이 성립. 그러므로,  $c=1$ 와  $N=1$ 을 선택하면,  $n \in O(n^2)$ .

- $n^3 \in O(n^2)$  ?

$n \geq N$ 인 모든  $n$ 에 대해서  $n^3 \leq c \times n^2$  이 성립하는  $c$ 와  $N$  값은 존재하지 않는다. 즉, 양변을  $n^2$ 으로 나누면,  $n \leq c$  가 되는데  $c$ 를 아무리 크게 잡더라도 그 보다 더 큰  $n$ 이 존재한다. 그러므로  $n^3 \notin O(n^2)$



$$O(n^2)$$

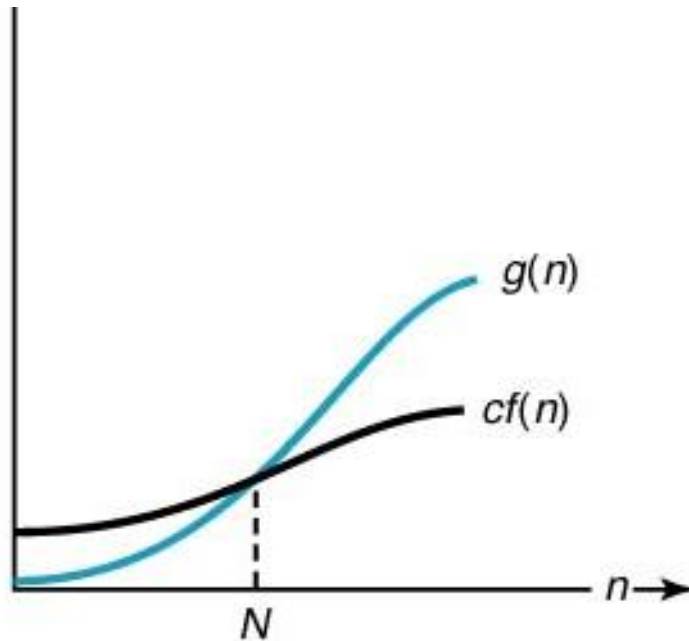


(a)  $O(n^2)$

# Ω 표기법

- 정의 : 점근적 하한(asymptotic lower bound)
  - ✓ 주어진 복잡도 함수  $f(n)$ 에 대해서  $g(n) \in \Omega(f(n))$ 이면  
 $n \geq N$ 인 모든 정수  $n$ 에 대해서  $g(n) \geq c \times f(n) \geq 0$ 이 성립하는 양의 실수  $c$ 와 음이 아닌 정수  $N$ 이 존재한다.
  - ✓  $\Omega(f(n)) = \{g(n): \text{there exist positive constants } c \text{ and } N \text{ such that } g(n) \geq c \times f(n) \geq 0 \text{ for all } n \geq N\}$
- $g(n) \in \Omega(f(n))$  읽는 방법:
  - $g(n)$ 은  $f(n)$ 의 오메가(omega)

## $\Omega$ 표기법



(b)  $g(n) \in \Omega(f(n))$

# Ω 표기법

- 어떤 함수  $g(n)$ 이  $\Omega(n^2)$ 에 속한다는 말은
  - ✓ 그 함수는  $n$ 이 커짐에 따라 (즉, 어떤 임의의  $N$  값보다 큰 값에 대해서는) 어떤 2차 함수  $cn^2$ 의 값보다는 **큰** 값을 가지게 된다는 것을 뜻한다. (그래프 상에서는 위에 위치)
  - ✓ 그 함수  $g(n)$ 은 어떤 2차 함수  $cn^2$  보다는 궁극적으로 **나쁘다**고 (느리다) 말할 수 있다.
- 어떤 알고리즘의 시간복잡도가  $\Omega(f(n))$ 이라면,
  - ✓ 입력의 크기  $n$ 에 대해서 이 알고리즘의 수행시간은 **아무리 빨라도**  $cf(n)$ 밖에 되지 않는다. ( $cf(n)$ 이 점근적하한이다.)
  - ✓ 이 알고리즘의 수행시간은  $cf(n)$ 보다 절대로 더 빠를 수는 없다.

## $\Omega$ 표기법 : 예

- $n^2 + 10n \in \Omega(n^2)$  ?

$n \geq 0$ 인 모든 정수  $n$ 에 대해서  $n^2 + 10n \geq n^2$  이 성립한다. 그러므로,  $c = 1$ 와  $N = 0$ 을 선택하면,  $n^2 + 10n \in \Omega(n^2)$ .

- $5n^2 \in \Omega(n^2)$  ?

$n \geq 0$ 인 모든 정수  $n$ 에 대해서,  $5n^2 \geq 1 \times n^2$  이 성립한다. 그러므로,  $c = 1$ 와  $N = 0$ 을 선택하면,  $5n^2 \in \Omega(n^2)$ .

## $\Omega$ 표기법 : 예 (계속)

- $T(n) = \frac{n(n-1)}{2}$  ?

$n \geq 2$ 인 모든  $n$ 에 대해서  $n-1 \geq \frac{n}{2}$  이 성립한다. 그러므로,  
 $n \geq 2$ 인 모든  $n$ 에 대해서  $\frac{n(n-1)}{2} \geq \frac{n}{2} \times \frac{n}{2} = \frac{1}{4}n^2$  이 성립한다.

따라서  $c = \frac{1}{4}$  과  $N = 2$ 를 선택하면,  $T(n) \in \Omega(n^2)$

- $n^3 \in \Omega(n^2)$  ?

$n \geq 1$ 인 모든 정수  $n$ 에 대해서,  $n^3 \geq 1 \times n^2$  이 성립한다.  
그러므로,  $c = 1$ 와  $N = 1$ 을 선택하면.

$$n^3 \in \Omega(n^2)$$

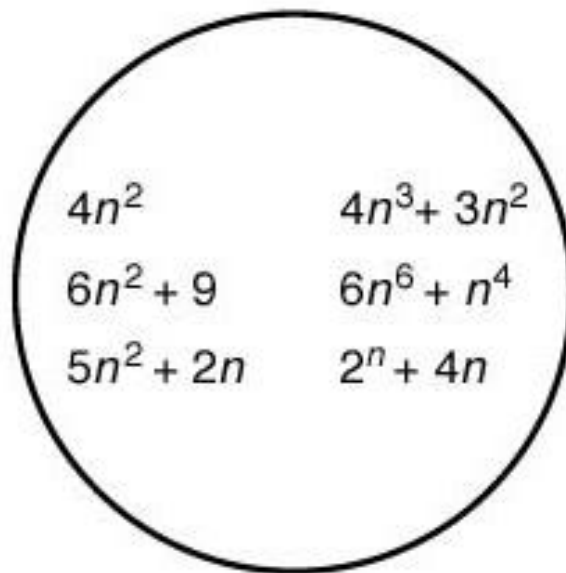
## $\Omega$ 표기법 : 예 (계속)

- $n \in \Omega(n^2)$  ?

모순유도에 의한 증명(proof by contradiction):

$n \in \Omega(n^2)$ 이라고 가정. 그러면  $n \geq N$ 인 모든 정수  $n$ 에 대해서,  $n \geq c \times n^2$ 이 성립하는 실수  $c > 0$ , 그리고 음이 아닌 정수  $N$ 이 존재한다. 위의 부등식의 양변을  $cn$ 으로 나누면  $\frac{1}{c} \geq n$ 가 된다. 그러나 이 부등식은 절대 성립할 수 없다. (왜냐하면  $n \geq N$ 인 모든 정수  $n$ 을 가정했는데,  $n$ 은 특정값보다 작아야 한다는 조건이 생겼음) 따라서 위의 가정은 모순이다. 그러므로  $n \notin \Omega(n^2)$

$$\Omega(n^2)$$



(b)  $\Omega(n^2)$



## ⊖ 표기법

- 정의 : (asymptotic tight bound)

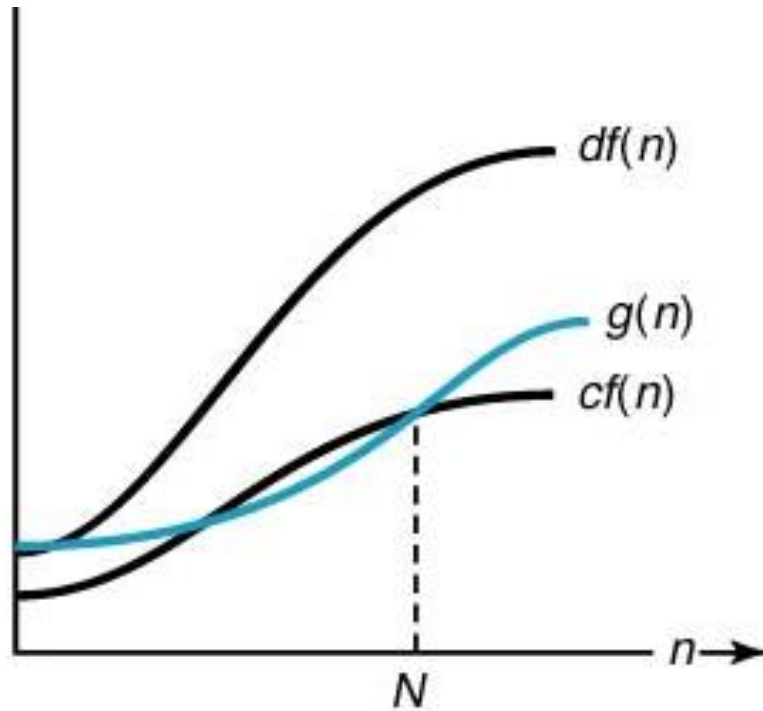
- ✓ 복잡도 함수  $f(n)$  에 대해서  $\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$
- ✓  $n \geq N$  인 모든 정수  $n$  에 대해서  $c \times f(n) \leq g(n) \leq d \times f(n)$  이 성립하는 양의 실수  $c$ 와  $d$ , 음이 아닌 정수  $N$ 이 존재한다.
- ✓  $\Theta(f(n)) = \{g(n) : \text{there exist positive constants } c, d, \text{ and } N \text{ such that } c \times f(n) \leq g(n) \leq d \times f(n) \text{ for all } n \geq N\}$

- 참고:  $g(n) \in \Theta(f(n))$ 은 “ $g(n)$ 은  $f(n)$ 의 차수(order)”라고 한다.

- 예 :  $T(n) = \frac{n(n-1)}{2}$  은  $O(n^2)$  이면서  $\Omega(n^2)$ 이다. 따라서

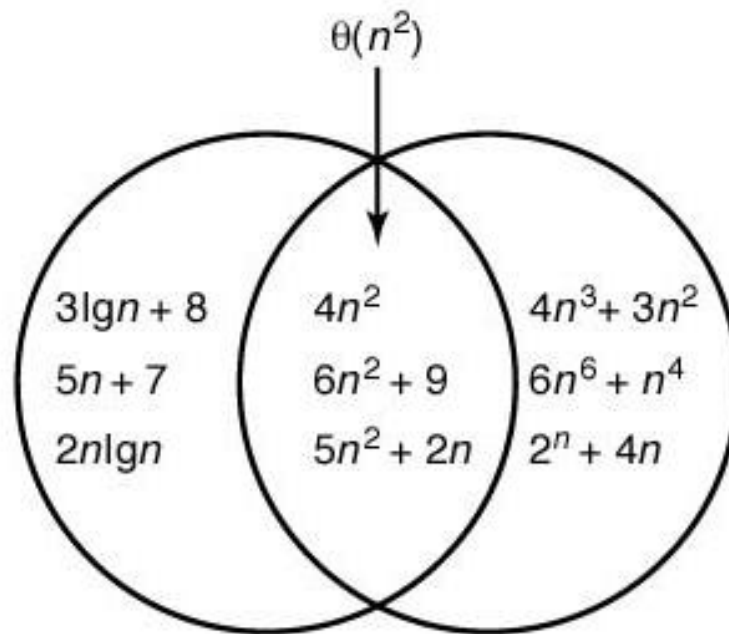
$$T(n) \in \Theta(n^2)$$

## ⊕ 표기법



(c)  $g(n) \in \Theta(f(n))$

$$\Theta(n^2)$$



$$(c) \theta(n^2) = O(n^2) \cap \Omega(n^2)$$

# 작은(small) $o$ 표기법

- 작은  $o$ 는 복잡도 함수끼리의 관계를 나타내기 위한 표기법이다.

- 정의 : 작은  $o$

주어진 복잡도 함수  $f(n)$ 에 대해서  $g(n) \in o(f(n))$ 이면

모든 양의 실수  $c$ 에 대해,  $n \geq N$ 인 모든  $n$ 에 대해서  $0 \leq g(n) \leq c \times f(n)$  이 성립하는 음이 아닌 정수  $N$ 이 존재한다.

✓  $o(f(n)) = \{g(n) : \text{for any positive constants } c > 0, \text{ there exists a constant } N > 0 \text{ such that } 0 \leq g(n) \leq c \times f(n) \text{ for all } n \geq N\}$

- 참고:  $g(n) \in o(f(n))$ 은 “ $g(n)$ 은  $f(n)$ 의 작은 오( $o$ )”라고 한다.

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$$

## 큰 $O$ vs. 작은 $o$

- 큰  $O$ 와의 차이점
  - 큰  $O$  - 실수  $c > 0$  중에서 하나만 성립하여도 됨
  - 작은  $o$  - **모든** 실수  $c > 0$ 에 대해서 성립하여야 함
- $g(n) \in o(f(n))$ 은  $g(n)$  이 궁극적으로  $cf(n)$ 보다 훨씬 낮다(좋다, 작은 값)는 의미이다.

## 작은 $o$ 표기법 : 예

- $n \in o(n^2)$  ?

증명:

- $c > 0$ 이라고 하자.  $n \geq N$ 인 모든  $n$ 에 대해서  $n \leq cn^2$  이 성립하는  $N$  을 찾아야 한다.
- 이 부등식의 양변을  $cn$ 으로 나누면  $1/c \leq n$  을 얻는다.
- 따라서  $N \geq 1/c$ 가 되는 어떤  $N$  을 찾으면 된다.
- 여기서  $N$  의 값은  $c$ 에 의해 좌우된다.
- 예를 들어 만약  $c=0.0001$  이라고 하면,  $N$  의 값은 최소 10,000이 되어야 한다.
- 즉,  $n > 10,000$ 인 모든  $n$ 에 대해서  $n \leq 0.0001n^2$  이 성립한다.

## 작은 $o$ 표기법 : 예 (계속)

- $n \in o(5n)$  ?

모순 유도에 의한 증명:

- $c=1/6$  이라고 하자.
- $n \in o(5n)$ 이라고 가정하면,  $n \geq N$ 인 모든 정수  $n$ 에 대해서,  $n \leq 1/6 \times 5n = 5/6n$ 이 성립하는 음이 아닌 정수  $N$ 이 존재해야 한다.
- 그러나 그런  $N$ 은 절대로 있을 수 없다.
- 따라서 위의 가정은 모순이다.

## $\omega$ 표기법

- By analogy,  $\omega$  –notation is to  $\Omega$ -notation as  $o$ -notation is to  $O$ -notation.
- It denotes a lower bound that is not asymptotically tight.
  - ✓  $g(n) \in \omega(f(n))$  if and only if  $f(n) \in o(g(n))$
  - ✓  $\omega(f(n))$ : small omega of  $f$  of  $n$
  - ✓  $\omega(f(n)) = \{ g(n) : \text{for any positive constant } c > 0, \text{ there exists a constant } N > 0 \text{ such that } 0 \leq c \times f(n) \leq g(n) \text{ for all } n \geq N \}$
- $n^2/2 \in \omega(n)$ . But  $n^2/2 \notin \omega(n^2)$ .
- The relation  $g(n) \in \omega(f(n))$  implies that

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \infty$$



# 차수의 주요 성질 I

1.  $g(n) \in O(f(n))$  if and only if  $f(n) \in \Omega(g(n))$
2.  $g(n) \in \Theta(f(n))$  if and only if  $f(n) \in \Theta(g(n))$
3.  $b > 1$ 이고  $a > 1$ 이면,  $\log_a n \in \Theta(\log_b n)$ 은 항상 성립. 즉, 로그(logarithm) 복잡도 함수는 모두 같은 카테고리에 속한다. 따라서 통상  $\Theta(\lg n)$ 으로 표시한다.
4.  $b > a > 0$ 이면,  $a^n \in o(b^n)$ . 즉, 지수(exponential) 복잡도 함수가 모두 같은 카테고리 안에 있는 것은 아니다.

## 차수의 주요 성질 II

5.  $a > 0$ 인 모든  $a$ 에 대해서,  $a^n \in o(n!)$ . 다시 말하면,  $n!$ 은 어떤 지수 복잡도 함수보다도 나쁘다.
6. 복잡도 함수를 다음 순으로 나열해 보자.

$$\Theta(\lg n), \Theta(n), \Theta(n \lg n), \Theta(n^2), \Theta(n^j), \Theta(n^k), \Theta(a^n), \Theta(b^n), \Theta(n!)$$

여기서  $k > j > 2$ 이고  $b > a > 1$ 이다. 복잡도 함수  $g(n)$ 이  $f(n)$ 을 포함한 카테고리의 왼쪽에 위치한다고 하면,  $g(n) \in o(f(n))$

7.  $c \geq 0, d > 0$ ,  $g(n) \in O(f(n))$ , 그리고  $h(n) \in \Theta(f(n))$  이면,

$$c \times g(n) + d \times h(n) \in \Theta(f(n))$$

# 극한(limit)을 이용하여 차수를 구하는 방법

- 정리 1.3

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \begin{cases} c > 0 & \text{이면 } g(n) \in \Theta(f(n)) \\ 0 & \text{이면 } g(n) \in o(f(n)) \\ \infty & \text{이면 } f(n) \in o(g(n)) \end{cases}$$

- 예제 1.24 : 다음이 성립함을 보이시오.

$$\frac{n^2}{2} \in o(n^3) \qquad \lim_{n \rightarrow \infty} \frac{n^2/2}{n^3} = \lim_{n \rightarrow \infty} \frac{1}{2n} = 0$$

- 예제 1.25 :  $b > a > 0$  일 때,  $a^n \in o(b^n)$

$$\lim_{n \rightarrow \infty} \frac{a^n}{b^n} = \lim_{n \rightarrow \infty} \left( \frac{a}{b} \right)^n = 0 \text{ 왜냐하면, } 0 < \frac{a}{b} < 1$$

- 정리 1.3에 의해  $a > 0$  이면 다음이 성립

$$a^n \in o(n!)$$

(1) if  $a \leq 1$ , it is trivial.

(2) if  $a > 1$ , for a large  $n$ ,

$$n! > \overbrace{\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n}{2} \right\rfloor \cdots \left\lfloor \frac{n}{2} \right\rfloor}^{n/2 \text{ 개}} > a^4 a^4 \cdots a^4$$

$$\therefore \frac{a^n}{n!} < \frac{a^n}{a^4 a^4 \cdots a^4} = \frac{a^n}{(a^4)^{n/2}} = \frac{a^n}{a^{2n}} = \left(\frac{1}{a}\right)^n$$

$$\lim_{n \rightarrow \infty} \frac{a^n}{n!} = 0$$

$$\begin{array}{c} n/2 \text{ 개} \\ \underbrace{n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1}_{> n/2 \times n/2 \times \cdots \times n/2} \\ n/2 \text{ 개} \end{array}$$

## L'Hopital's rule

If

$$\lim_{x \rightarrow c} f(x) = \lim_{x \rightarrow c} g(x) = 0 \text{ or } \pm \infty$$

$$\lim_{x \rightarrow c} \frac{f'(x)}{g'(x)} \quad \text{exists, and}$$

$$g'(x) \neq 0 \quad \text{for all } x \text{ in } I(\text{interval}) \text{ with } x \neq c$$

then

$$\lim_{x \rightarrow c} \frac{f(x)}{g(x)} = \lim_{x \rightarrow c} \frac{f'(x)}{g'(x)}$$

## ● 정리 : 로피탈(L'Hopital)의 법칙

$$\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) = \infty \text{ 이면}$$

$$\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = \lim_{n \rightarrow \infty} \left( \frac{g'(n)}{f'(n)} \right) \text{ 이다.}$$

예제 1.27

$$\lg n \in o(n) \qquad \lim_{n \rightarrow \infty} \frac{\lg n}{n} = \lim_{n \rightarrow \infty} \left( \frac{\frac{1}{n \ln 2}}{1} \right) = 0$$

예제 1.28

$$\log_a n \in \Theta(\log_b n) \quad b > 1 \text{ and } a > 1$$

$$\lim_{n \rightarrow \infty} \frac{\log_a n}{\log_b n} = \lim_{n \rightarrow \infty} \left( \frac{\frac{1}{n \ln a}}{\frac{1}{n \ln b}} \right) = \frac{\log b}{\log a} > 0$$

# 알고리즘 복잡도와 컴퓨터 능력

1. 알고리즘의 복잡도가  $cn$ 일 때  $t$ 시간 동안

- ✓ 현재의 기계가 문제크기  $m$ 개의 문제 해결
- ✓ 기계의 처리 속도가 2배 된다면 문제크기  $2m$ 개의 문제를  $t$ 시간에 해결할 수 있다.

2. 알고리즘의 복잡도가  $cn^2$ 일 때  $t$ 시간 동안

- ✓ 현재의 기계가 문제크기  $m$ 개의 문제 해결
- ✓ 기계의 처리 속도가 4배 된다면 문제크기  $2m$ 개의 문제를  $t$ 시간에 해결할 수 있다.

3. 알고리즘의 복잡도가  $c2^n$ 일 때  $t$ 시간 동안

- ✓ 현재의 기계가 문제크기  $m$ 개의 문제 해결
- ✓ 기계의 처리 속도가 2배 된다면 문제크기  $m+1$ 개의 문제를  $t$ 시간에 해결할 수 있다.
- ✓ 기계의 처리 속도가 100배 된다면 문제크기 ??개의 문제를  $t$ 시간에 해결할 수 있다.

# 알고리즘 복잡도와 컴퓨터 능력

- 알고리즘의 복잡도가 (A)일 때  $t$ 시간 동안
  - ✓ 현재의 기계가 문제크기  $m$ 개의 문제 해결
  - ✓ 기계의 처리 속도가 (B)배 된다면 문제크기 (C)개의 문제를  $t$ 시간에 해결할 수 있다.

A	B	C
$cn$	2	$2m$
$cn^2$	4	$2m$
$cn^3$	8	$2m$
$c2^n$	2	$m+1$
$c2^n$	100	?

$t$  시간에  $cm^3$  만큼의 연산 가능하므로

기계의 성능이 8배 되었으므로  $t$  시간에  $8cm^3$  만큼의 연산 가능. 증가된 문제크기를  $x$ 라 하면

$$c(m+x)^3 = 8cm^3$$

$x = m$ , 총 문제크기는  $2m$





1장 끝

수고하셨습니다.