



# 알고리즘



## 알고리즘 소개

2012 - 1

**Myoung-Jin Kim, Ph.D. (webzealer@ssu.ac.kr)**  
**Research Professor(Soongsil University)**

1강

# 알고리즘 소개

- ❖ 알고리즘의 기본 개념
- ❖ 기본 자료구조
- ❖ 알고리즘의 설계 및 분석
- ❖ 점근 성능
- ❖ 순환 알고리즘의 성능

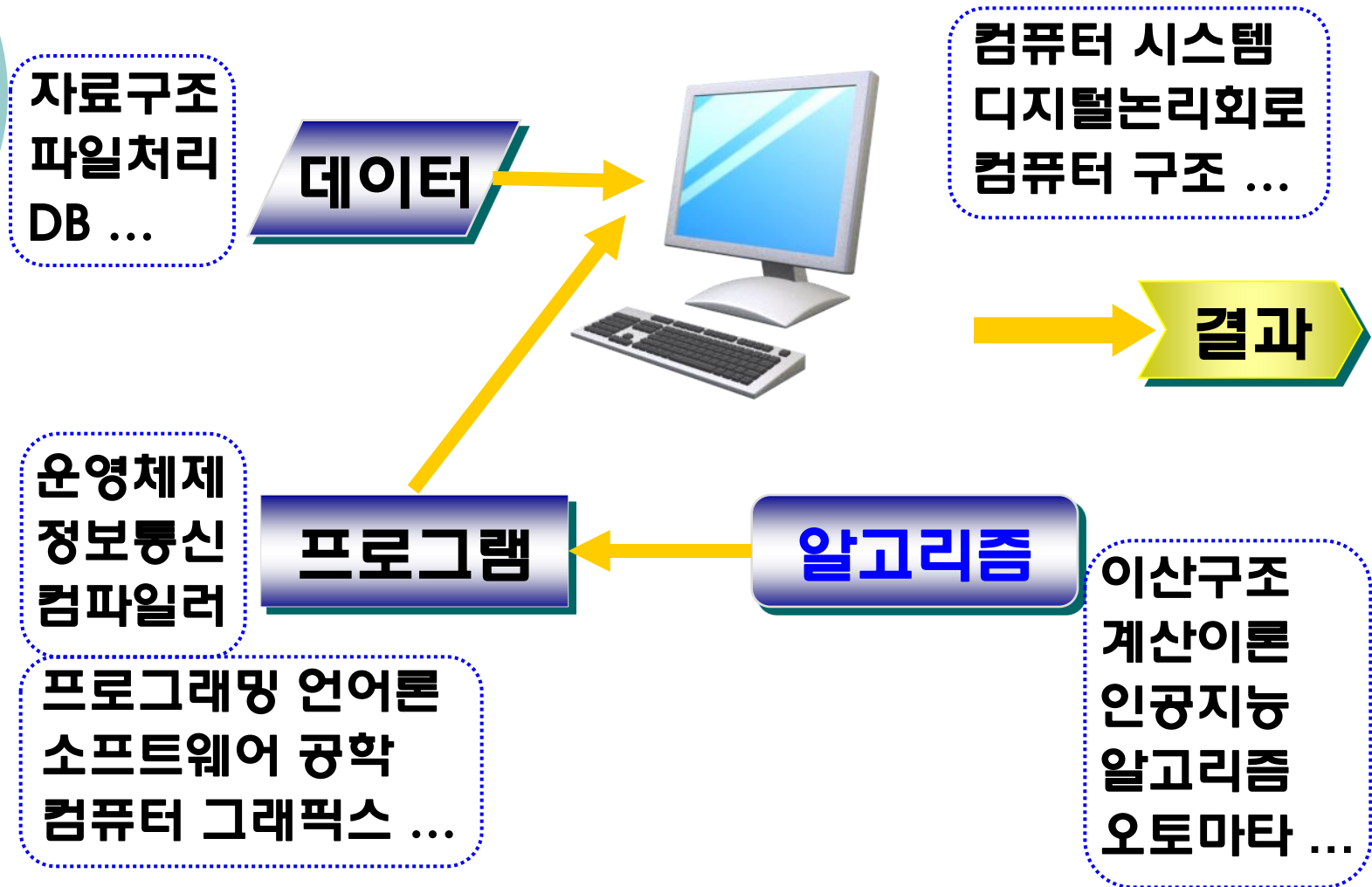


컴퓨터학과 이관용 교수

알고리즘

# 알고리즘의 기본 개념

# 알고리즘의 중요성



# 알고리즘이란?



주어진 문제를 풀기 위한 명령어들을 단계적으로 나열한 것

문제를 해결하거나 함수를 계산하기 위해 쫓아야 할 **모호함이 없는 간단한 명령들로 구성된 일련의 순서적 단계**

“an ordered set of unambiguous, executable steps that produces a result and terminates in a finite time”

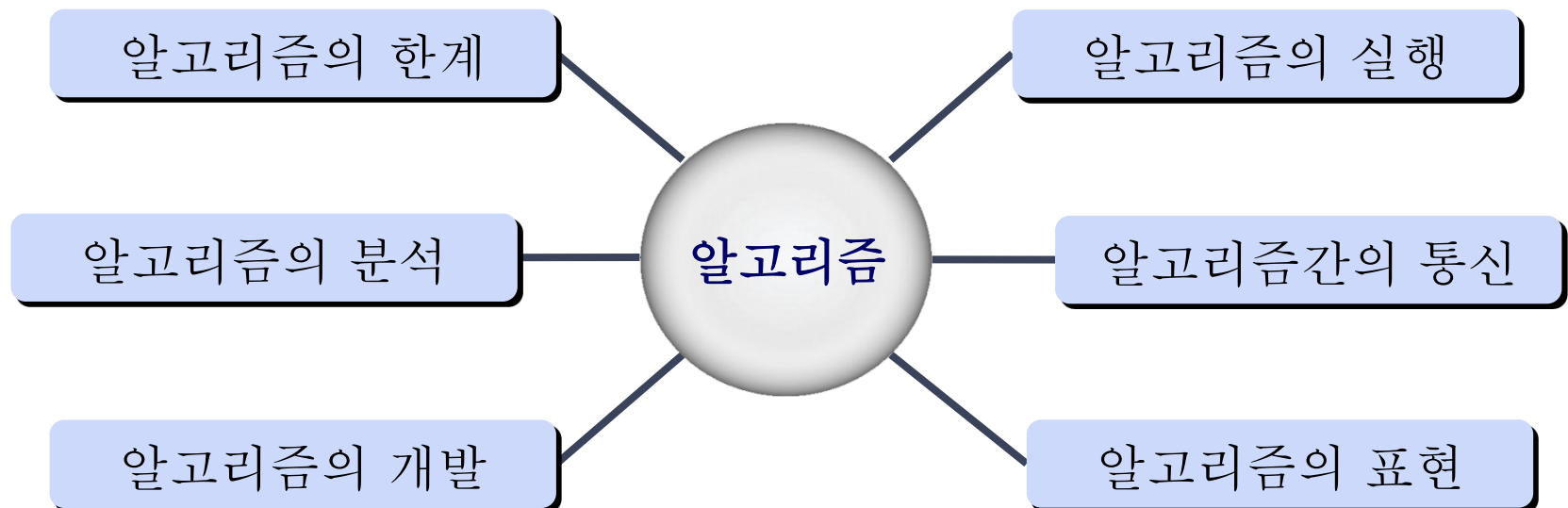
튜링 기계에 의해 수행 가능한 프로시저

→ **계산 이론** (computational theory)



# 알고리즘의 중요성

- 컴퓨터의 한계
  - 알고리즘의 존재 여부
- “컴퓨터과학 = 알고리즘 과학”



# 알고리즘의 정의와 요건

---

- ✓ **입출력**

- 0개 이상의 외부 입력
- 1개 이상의 출력

- ✓ **모호하지 않고 단순 명확한 명령**

- ✓ **한정된 수의 작업 후에는 반드시 종료**

- ✓ **모든 명령은 수행 가능해야 함**

- ✓ **(실용적 관점) 효율적이어야 함**

**주어진 문제에 대한 결과를 생성하기 위해  
모호하지 않고 간단하며 컴퓨터가 수행 가능한  
일련의 유한개의 명령들을 순서적으로 구성한 것**

# 알고리즘 생성 단계

---

## 설 계

상향식 설계  
하향식 설계

## 표현/기술

일상 언어, 순서도,  
의사 코드, 프로그래밍 코드 등

## 정확성 검증

수학적 검증  
실용적 검증

## 효율성 분석

공간 복잡도  
시간 복잡도



# 알고리즘 기술 언어

---

- 알고리즘 기술 방법
  - 자연어에 의한 표현
    - 일상 언어 - 한글 또는 영문
    - 의사 코드(Pseudo Code)
  - 도형에 의한 표현
    - 순서도(흐름도:Flow Chart)
    - 박스(Box) 다이어그램
- C언어에 기반한 의사 프로그래밍 언어

# 알고리즘 기술 언어 (C-유사)

---

## 변수선언

```
int min, max;  
float degree, b;  
char ch, token;
```

## 지정문

```
min = 10;  
degree = a + b;
```

## 반복문

```
while (age <= adult) {  
    ...  
}
```

```
for (i=1; i < n; i++ ) {  
    ...  
}
```

# 알고리즘 기술 언어 (C-유사)

## 조건문

```
if ( age > adult )  
    문장1  
else 문장2
```

## 배열

```
int num[10];  
num[0], ..., num[9]
```

## 함수

```
Power (x, n)  
{  
    p=1;  
    for (i=1; i<=n; i+ + )  
        p = p * x;  
    return (p);  
}
```

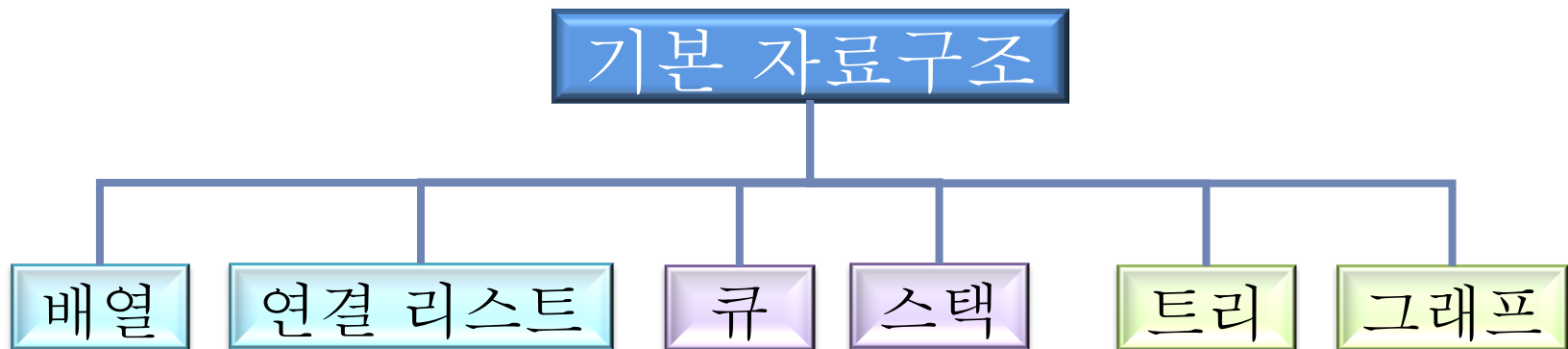
```
printf( Power (2, 5) );
```

알고리즘

# 기본 자료구조

# 자료구조와 알고리즘의 관계

- 자료구조
  - 컴퓨터 기억공간 내에 자료를 표현하고 조직화시키는 방법
- 자료구조의 선택과 알고리즘의 효율성의 관계
  - 자료구조 단순 → 연산 단계 및 수행 시간의 증가
  - 자료구조 복잡 → 연산 횟수 감소
- 프로그램 ← 자료구조 + 알고리즘



# 기본 자료구조

## ○ 배열

### 정적 구조

- 배열내의 각 원소 접근 시간이 동일하므로 원소들을 임의 순서로 처리할 경우 유리
- 배열의 중간에서의 삽입·삭제 시 많은 시간 소요

A	B	D	E	F	G	H	
---	---	---	---	---	---	---	--

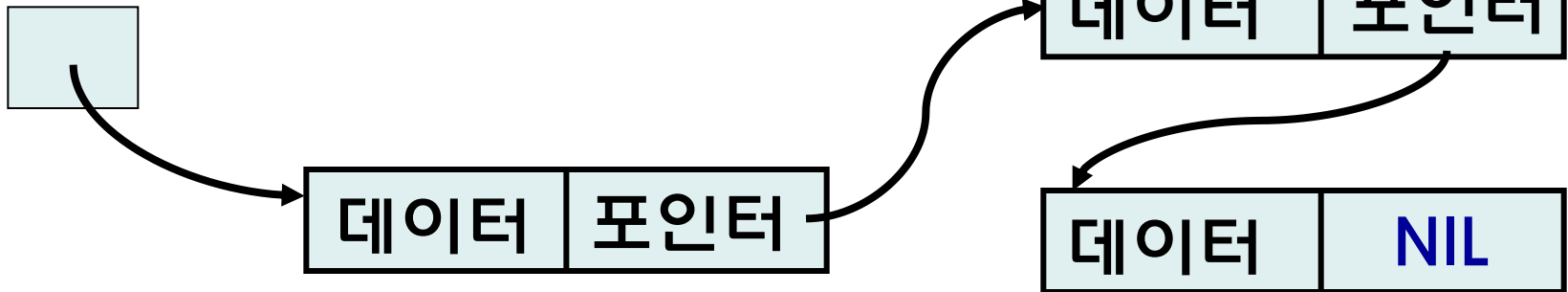
# 기본 자료구조

## ○ 연결 리스트

### 동적 구조

- 삽입과 삭제가 용이
- 리스트내의 한 노드를 찾기 위해 그 노드 앞에 위치하는 모든 노드를 차례대로 따라가야만 함

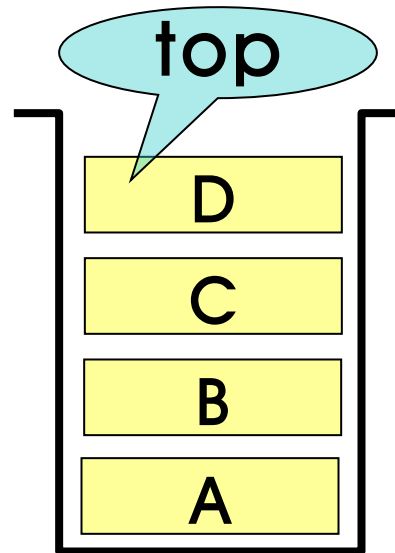
헤드 포인터



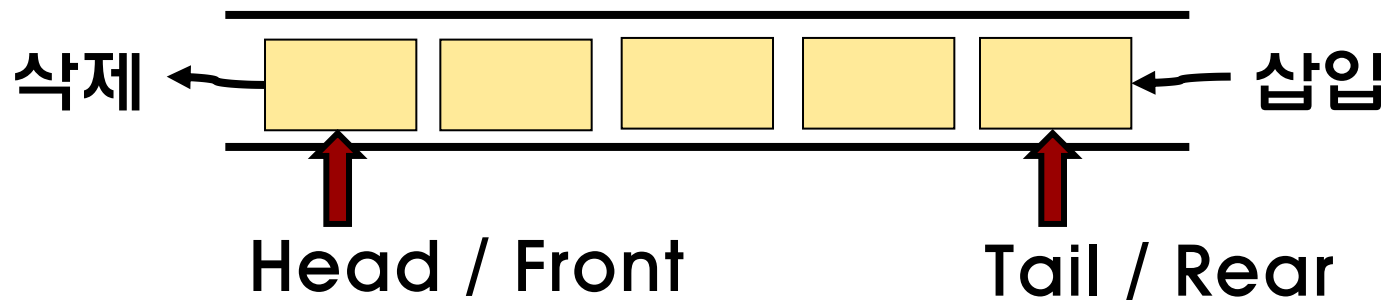


# 기본 자료구조

## ○ 스택

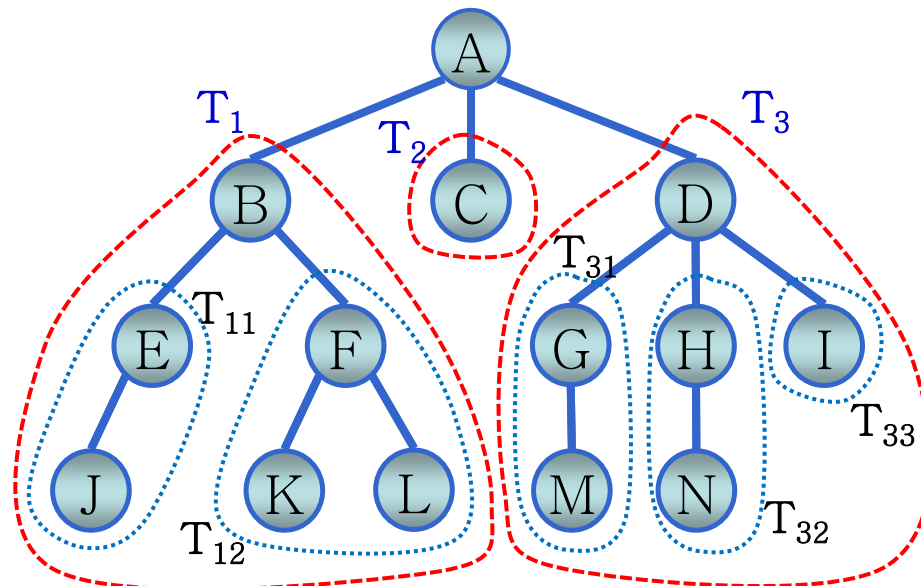


## ○ 큐



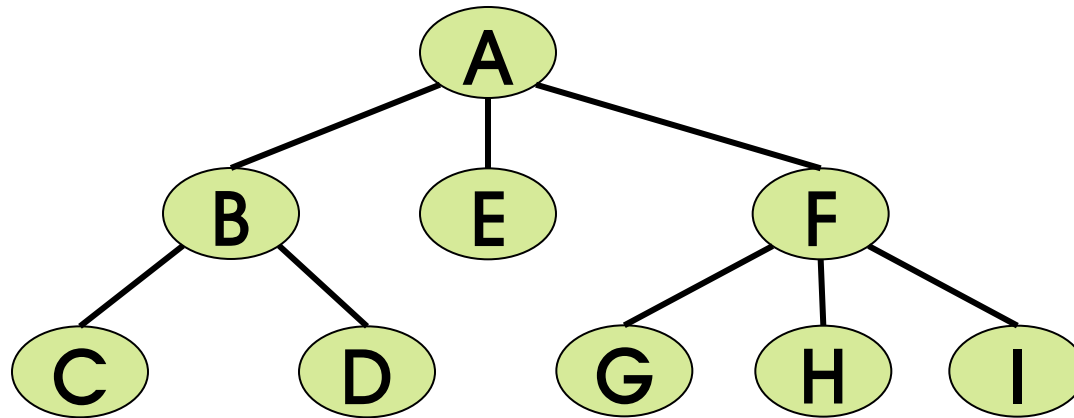
# 트리

- 하나 이상의 노드로 구성된 유한 집합  $T$ 
  - $T$ 의 원소 가운데 단 하나의 루트 노드가 존재
  - 루트 노드를 제외한 나머지 노드는  $n$ 개( $n \geq 0$ )의 서로 분리된 부분집합  $T_1, T_2, \dots, T_n$  (“서브트리”)으로 나누어진다.



# 트리

- 뿌리나무(rooted tree)
  - 나무의 정점 중 하나가 뿌리(root)로 지정된 것



# 기본 자료구조

---

뿌리가  $r$ 인 뿌리 나무  $T$ 의 어떤 노드  $x$ 에 대하여

- 조상(ancestor)과 자손(descendant)
  - $r$ 로부터  $x$ 까지의 경로상에 존재하는 노드  $y$ 는  $x$ 의 조상이고,  $x$ 는  $y$ 의 자손
- 부분 나무(subtree)
  - 뿌리  $x$ 와  $x$ 의 자손들로 이루어진 나무
- 부모(parent)
  - $x$ 의 바로 위쪽에 연결된 노드
- 자식(child)
  - $x$ 의 아래쪽에 연결된 노드
- 동기(sibling)
  - 부모가 같은 노드들
- 잎(leaf)
  - 자식이 없는 노드

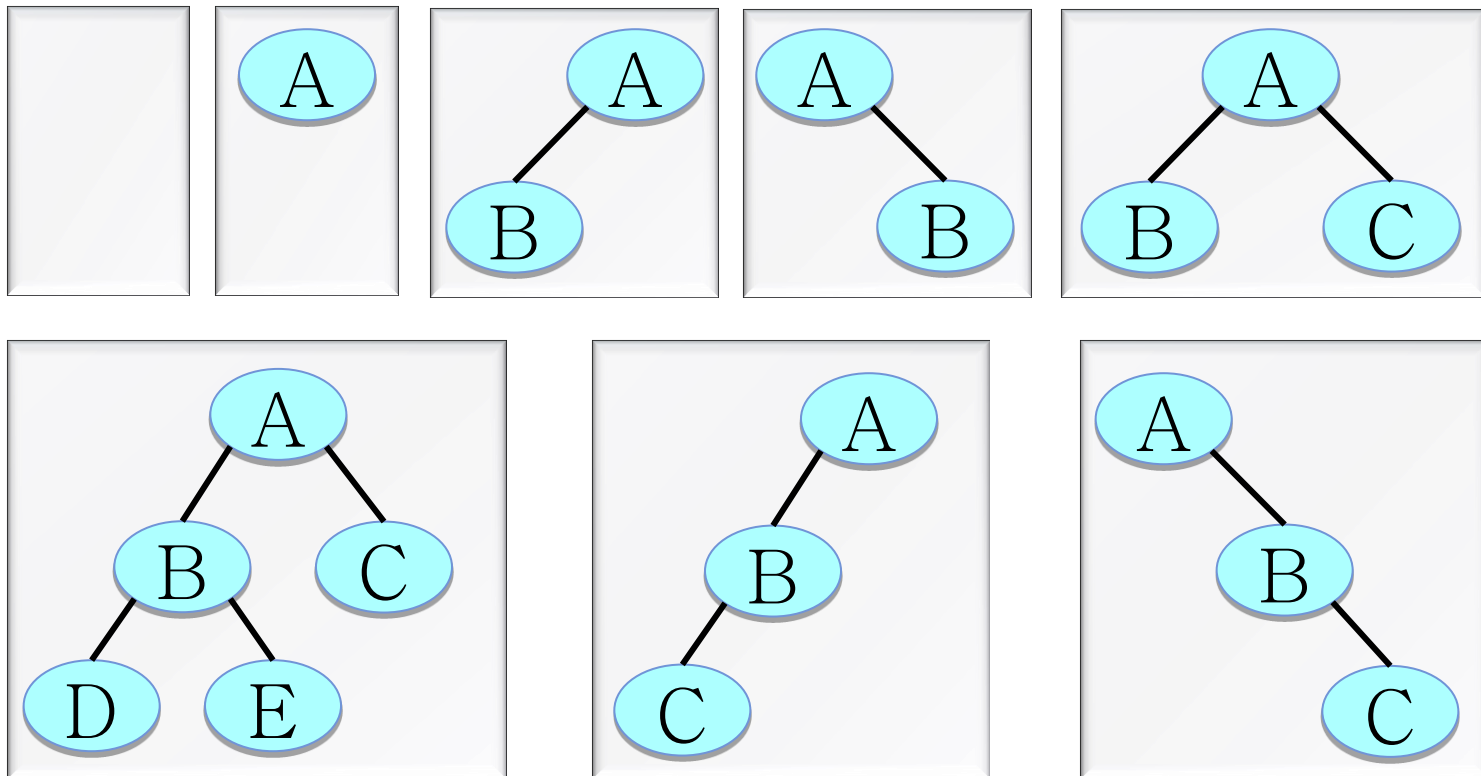
# 기본 자료구조

---

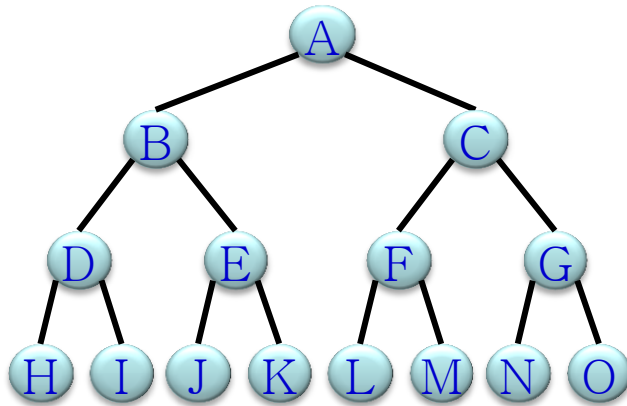
- 차수(degree)
  - 뿌리 나무 T에서 노드 x의 자식의 개수
- 깊이(depth)
  - 뿌리 r에서 노드 x까지의 경로의 길이
- 높이(height)
  - 뿌리 나무 T에서 가장 큰 깊이
- 수준(level)
  - 깊이가 같은 노드들을 동일한 수준에 있는 노드들이라 함
- 순서 나무(ordered tree)
  - 노드의 각 자식에 순서가 부여된 뿌리 나무
- 이진 나무(binary tree)
  - 각 노드의 자식이 2개 이하인 순서 나무

# 이진 트리

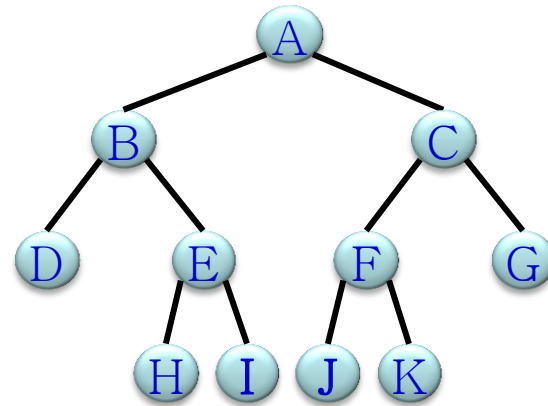
- 각 노드의 차수가 2 이하인 순서 트리



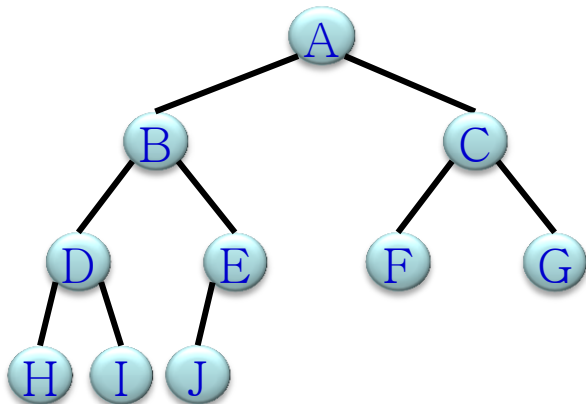
# 이진 트리의 종류



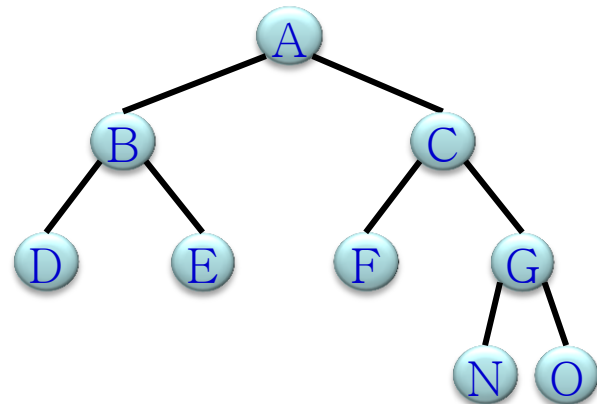
포화 perfect 이진 트리



전 full 이진 트리



완전 complete 이진 트리



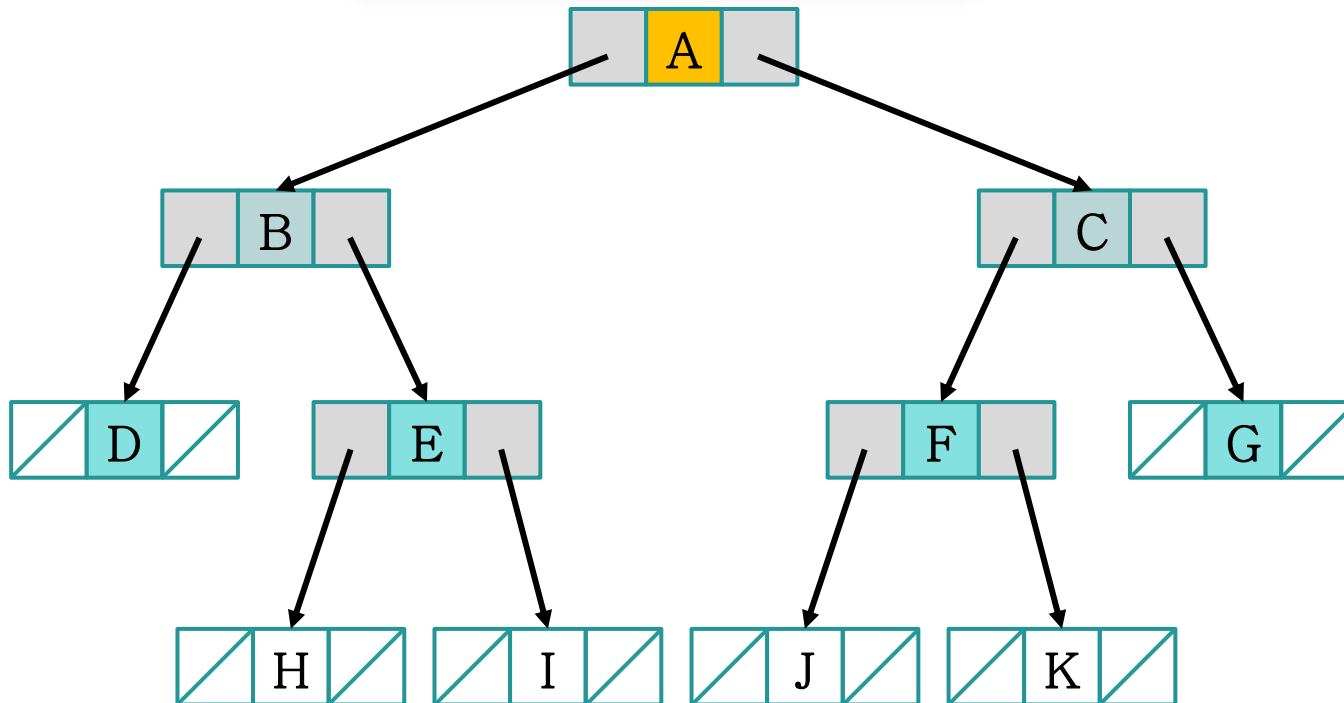
균형 balanced 이진 트리



# 이진 트리의 구현



일차원 배열을 이용한 구현



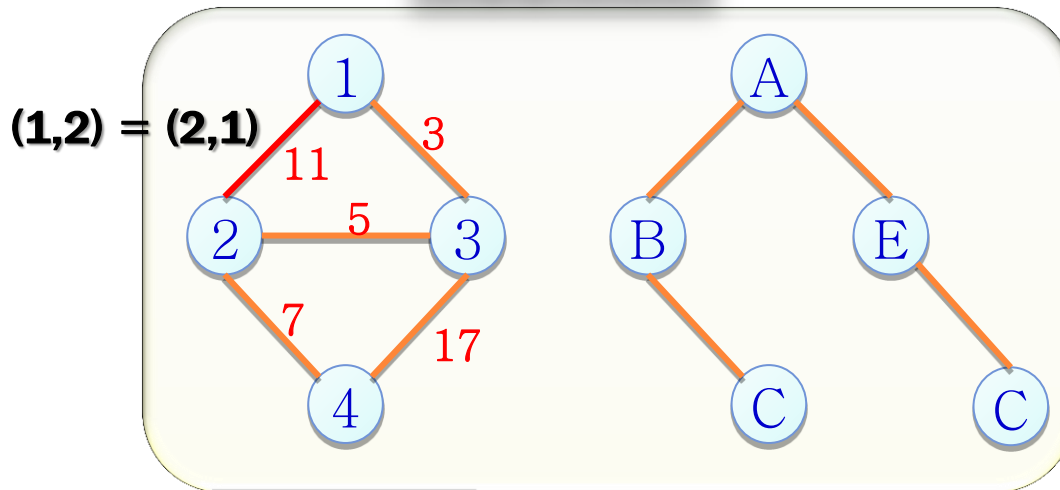
연결 리스트를 이용한 구현

# 그래프

○ 그래프  $G=(V,E)$

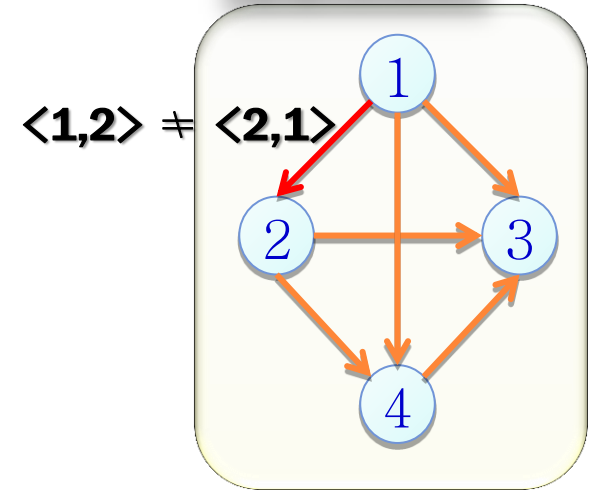
- $V$  : 정점의 집합,  $E$  : 간선의 집합

무방향 그래프



가중 그래프

방향 그래프

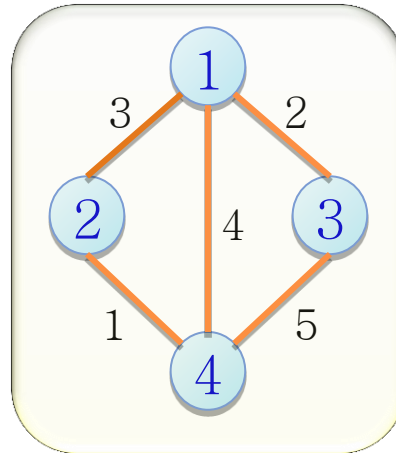


# 그래프

## ○ 주요 용어

- 인접 adjacent, 부속 incident
- 부분 그래프 subgraph
- 경로 path, 경로의 길이 length
- 차수 degree
  - 진입차수 in-degree, 진출차수 out-degree
- 단순 경로 simple path, 사이클 cycle, 루프 loop
- 연결 connected
- 강력 연결 strongly connected

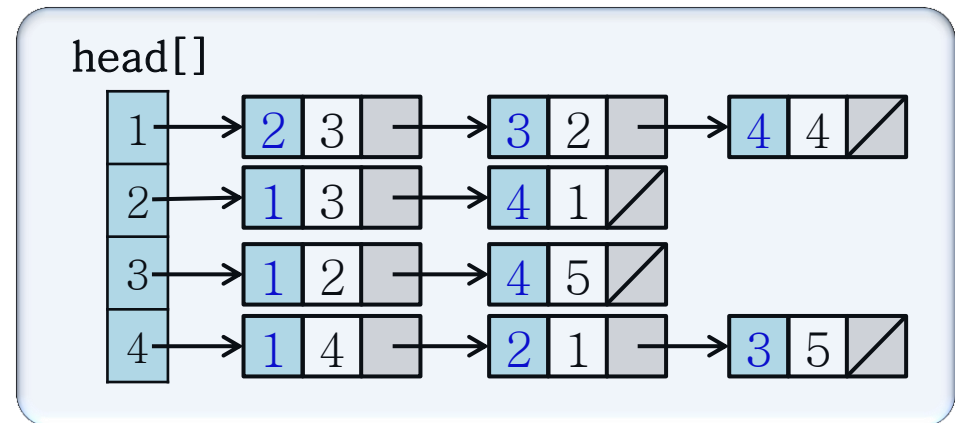
# 그래프의 구현



인접 행렬

	1	2	3	4
1	0	3	2	4
2	3	0	$\infty$	1
3	2	$\infty$	0	5
4	4	1	5	0

인접 리스트

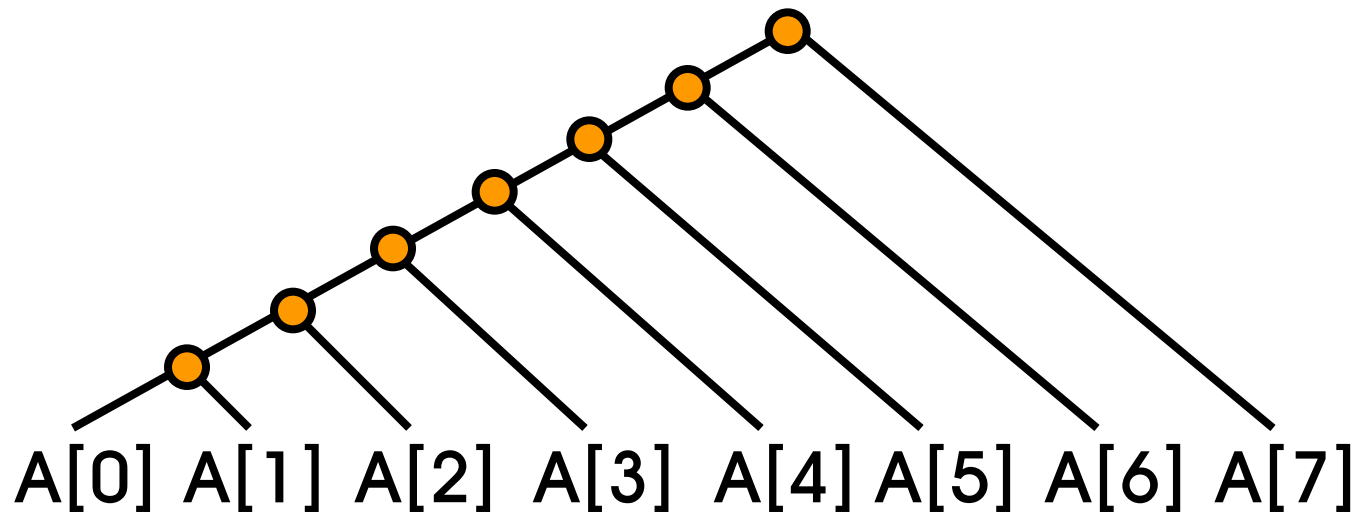


알고리즘

# 알고리즘의 설계

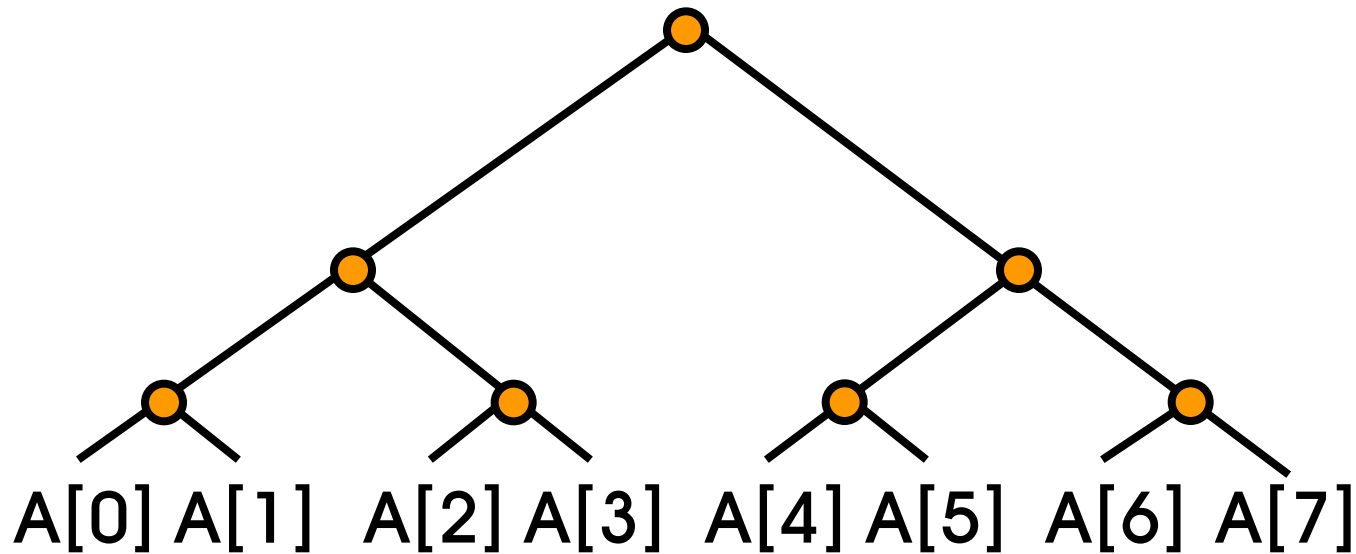
# 알고리즘 설계 → “창조적 활동”

## ○ 최대값 찾기 : 방법1



# 알고리즘 설계

## ○ 최대값 찾기 : 방법2

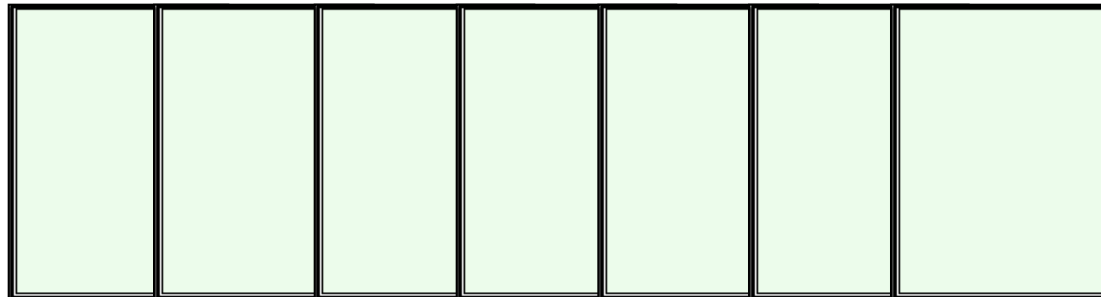
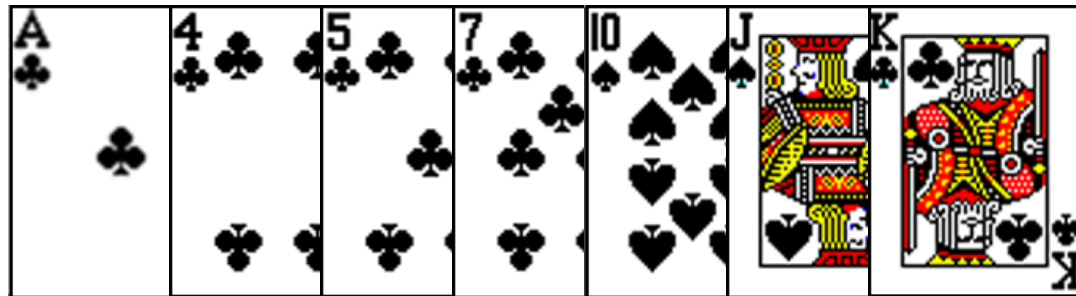


### 방법1 & 2

→ 가장 작은 횟수의 비교( $n-1$ 번)로 최대수를 찾는 효율적인 알고리즘

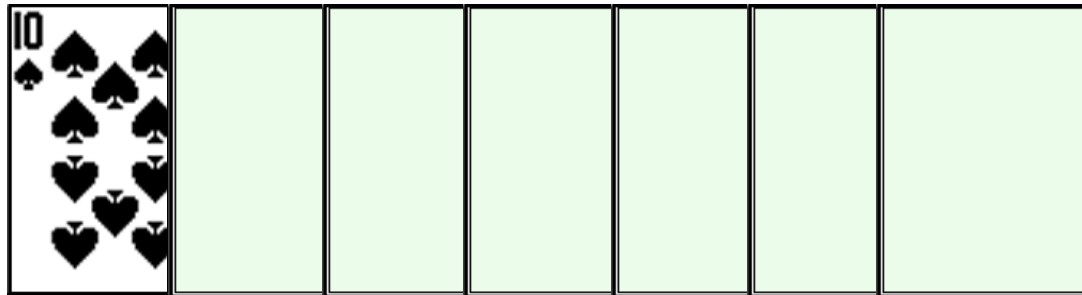


# 뒤섞인 카드에서 K 찾기

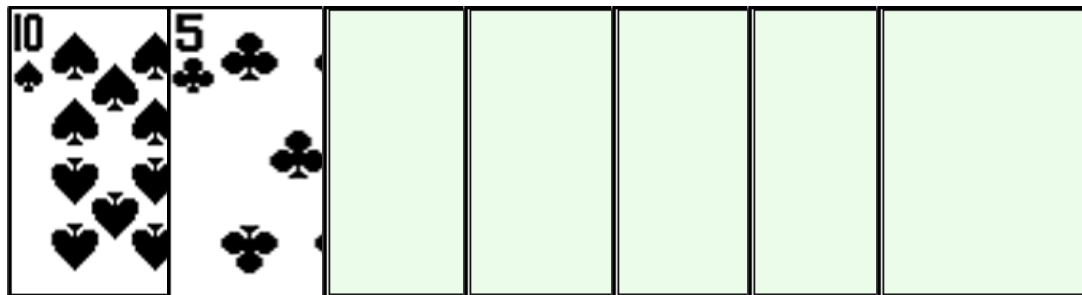


# 뒤섞인 카드에서 K 찾기

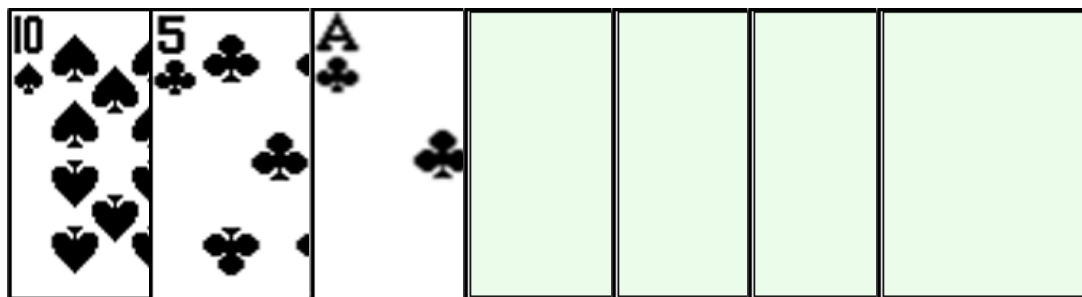
(1)



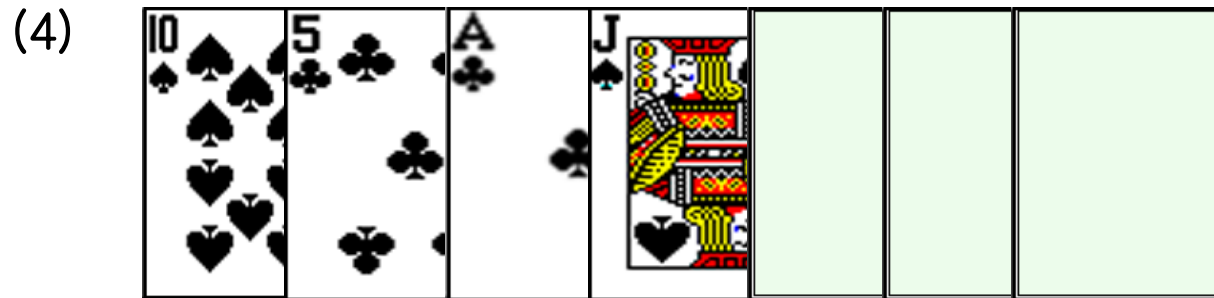
(2)



(3)



# 뒤섞인 카드에서 K 찾기



# 순차 탐색

---

**SeqSearch (A, n, x)**

**/\* 입력 : A, n, x**

**출력 : j    \*/**

**1    {**

**2        j = 1;**











**3        while (j ≤ n   &&   A(j) ≠ x)**

**4            j = j + 1;**

**5        return (j);**

**6    }**

# 순서대로 나열된 카드에서 J 찾기

①							
②							
③							
④							

“이진 탐색”

# 이진 탐색 알고리즘

---

BinarySearch (A[], key, left, right)

// A[mid]=key인 인덱스 mid를 반환

{

1 mid = (left + right)/2;

2 if (A[mid] = key) return(mid);

3 else if (A[mid] > key) BinarySearch(A, key, left, mid-1)

4       else BinarySearch(A, key, mid+ 1, right);

}

# 알고리즘 설계

---

- 설계 방법은 다양하며, 일정한 틀을 제시하기 곤란
- 대표적인 설계 기법
  - 욕심쟁이 방법
    - greedy method
  - 분할 정복 방법
    - divide and conquer method
  - 동적 프로그래밍 방법
    - dynamic programming method



# 분할정복 방법

- 순환적으로 문제를 푸는 방법

분할

주어진 문제를 여러 개의 작은 문제로 분할

정복

작은 문제들을 순환적으로 처리하여 정복.  
만약 작은 문제의 크기가 충분히 작다면  
순환 호출 없이 작은 문제에 대한 해가  
구해짐

결합

작은 문제에 대해 정복된 해를 결합하여  
원래 문제의 해를 구함

- 2장(퀵 정렬, 합병 정렬), 3장(이진 탐색)

# 동적 프로그래밍 방법

---

- 최적화 문제의 해를 구하는 상향식 접근 방법
  - 주어진 문제를 여러 개의 부분 문제로 분할
  - 가장 작은 부분 문제부터 해를 구하여 테이블에 저장
  - 저장된 해를 이용하여 입력 크기가 보다 큰 원래 문제를 점진적으로 해결
- 4장. 모든 쌍 최단 경로를 찾는 플로이드 알고리즘
- 6장. 동적 프로그래밍 (연쇄 행렬 곱셈, 스트링 편집 거리)

# 욕심쟁이 방법

- 해를 구하는 일련의 선택 과정마다 그 단계에서 ‘**가장 최선**’이라고 여겨지는 국부적인 최적해를 선택해 나가면 결과적으로 전체적인 최적해를 구할 수 있을 것이라고 **희망적인** 전략을 취하는 방법
  - 희망적 → 각 단계마다 선택한 최적해가 전체 최적해를 만들어 내지 못할 수 있음을 의미
- 1장. 거스름돈 문제, 배낭 문제
- 4장. 프림 알고리즘, 크루스칼 알고리즘, 데이크스트라 알고리즘

# 욕심쟁이 방법의 한계

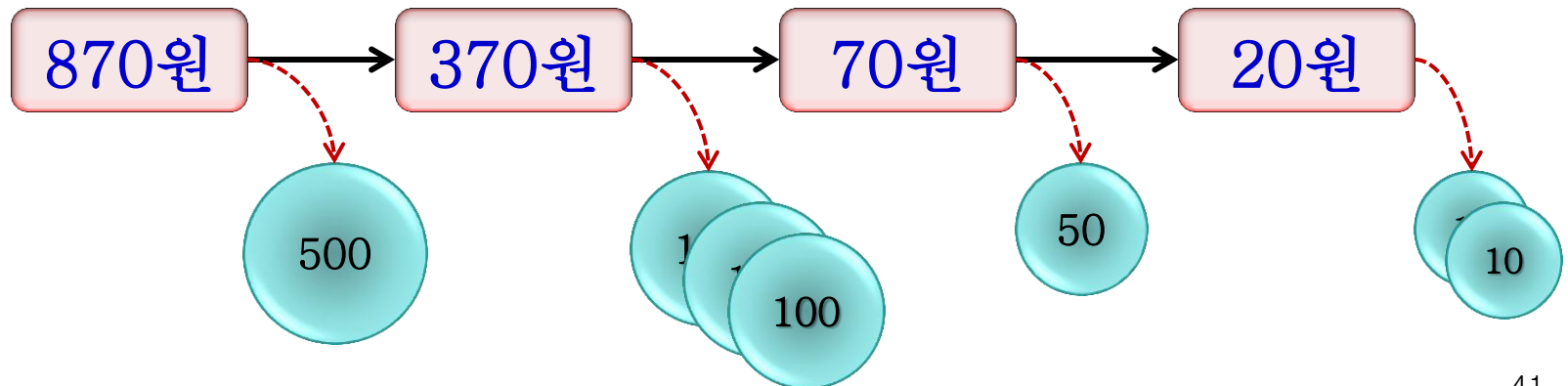
- 욕심쟁이가 세상에서 가장 멋진 상의와 바지, 세상에서 가장 멋진 넥타이를 구입하는 경우, 욕심쟁이는 세상에서 으뜸가는 멋쟁이?



- 욕심쟁이 방법으로 해를 구할 수 없는 문제도 있다.
- 그러나 욕심쟁이 방법을 적용할 수 있다면 아주 간단하게 알고리즘을 만들 수가 있다.

# 거스름돈 문제

- 고객에게 돌려줄 거스름돈이 T만큼 있을 때 고객이 받을 동전의 개수를 최소로 하면서 거스름돈을 돌려주는 문제
  - 동전의 종류 : 500원, 100원, 50원, 10원, 5원, 1원
- 단순히 액면가가 큰 동전부터 최대한 뽑아서 거스름돈을 제공



# 배낭 문제

---

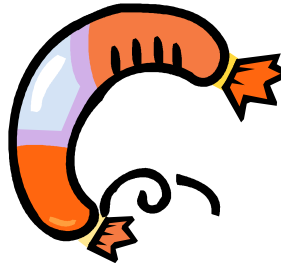
- 최대 용량  $M$ 인 하나의 배낭과 각각 무게  $w_i$ 와 이익  $p_i$ 가 부여되어 있는  $n$ 개의 물체가 있다고 가정
- 배낭 문제
  - 배낭의 용량을 초과하지 않는 범위에서 배낭에 들어있는 물체의 이익의 합이 최대가 되는 해를 찾아내는 것
  - 물체를 쪼갤 수 있다고 가정

# 배낭 문제



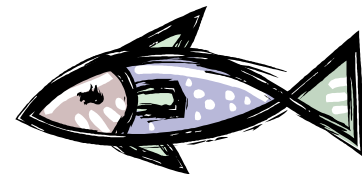
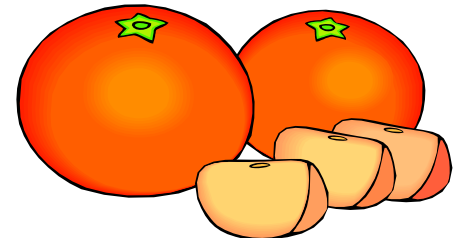
$M=10$     $n=4$

$$P_1 = 15$$
$$W_1 = 3$$



$$P_3 = 9$$
$$W_3 = 3$$

$$P_2 = 20$$
$$W_2 = 5$$



$$P_4 = 14$$
$$W_4 = 4$$

# 배낭 문제

- 문제

$$M = 10, \quad n = 4$$

$$(p_1, p_2, p_3, p_4) = (15, 20, 9, 14)$$

$$(w_1, w_2, w_3, w_4) = (3, 5, 3, 4)$$

- 각 물체의 가중치 (단위 무게당 이익)

$$\left( \frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \frac{p_4}{w_4} \right) = (5, 4, 3, 3.5)$$

- 최대 이익

$$\text{최대이익} = w_1 + w_2 + w_4 * \frac{1}{2} = 42$$



# 0/1 배낭 문제

- 물체를 쪼갤 수 없는 경우

$$M = 10, \quad n = 4$$

$$(p_1, p_2, p_3, p_4) = (15, 20, 9, 14)$$

$$(w_1, w_2, w_3, w_4) = (3, 5, 3, 4)$$

$$\left( \frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \frac{p_4}{w_4} \right) = (5, 4, 3, 3.5)$$

$$\text{최대이익} = W_1 + W_2 = 35$$

$$\text{실제 최대이익} = W_1 + W_3 + W_4 = 38$$

욕심쟁이 방법 적용 불가

알고리즘

# 알고리즘의 분석

# 알고리즘 분석

## ○ 정확성 분석

- 유효한 입력, 유한 시간, 정확한 결과 산출 여부

실용적



테스트 데이터에  
대한 디버깅결과

→ 프로그램이 정확하다는 것을 증명하지 못함

이론적



수학적 증명

→ 논리적으로 정확함을 보임

Vs.

# 알고리즘 분석

---

## ○ 효율 분석

- 알고리즘 수행에 필요한 컴퓨터 자원의 양을 측정
- 시간 복잡도 (time complexity)
  - 알고리즘이 필요로 하는 수행 시간의 크기
- 공간 복잡도 (space complexity)
  - 알고리즘이 필요로 하는 메모리 공간의 크기  
(정적 공간 + 동적 공간)

# 시간 복잡도

---

- 알고리즘의 수행 시간
  - 실제 실행 시간을 측정
    - 일반성 결여
  - 수행하는 기본 연산의 수행 회수로 계산

## 알고리즘의 수행시간

$$= \sum \{ \text{각 문장(연산)의 수행 회수} \\ \times \text{해당 문장의 수행 시간} \}$$

→ 모두 동일하다고 생각함

# 시간 복잡도

- 입력 크기의 함수로 표현

- 입력 크기 : 입력되는 데이터의 크기
  - 문제가 해결하고자하는 대상이 되는 개체의 개수
  - 행렬의 크기, 리스트 원소의 수, 그래프 정점의 수 등

- 입력 상태에 종속적  $A(n) = \sum_{I \in S_n} P(I)t(I)$

- 평균 수행 시간  $A(n)$

$$W(n) = \max_{I \in S_n} t(I)$$

- 최악 수행 시간  $W(n)$

$$B(n) = \min_{I \in S_n} t(I)$$

- 최선 수행 시간  $B(n)$

$S_n$  : 크기가  $n$ 인 입력들의 집합

$P(I)$  : 입력  $I$ 가 발생할 확률

$t(I)$  : 입력  $I$ 일 때 알고리즘의 수행시간

# 순차 탐색의 수행 시간

SeqSearch (A, n, x)

/\* 입력 : A, n, x      출력 : j \*/

1 {

2     j = 1;

3     while (j ≤ n && A(j) != x)

4         j = j + 1;

5     return (j);

6 }

$C_1$

$C_2$  k번

$C_3$  k-1번

$C_4$

$T(n) = 2n + 3$

$$c_1 + c_2k + c_3(k-1) + c_4 = (c_2 + c_3)k + (c_1 - c_3 + c_4) = ak + b$$

$O(n)$

# 순차 탐색의 수행 시간

$$c_1 + c_2k + c_3(k-1) + c_4 = (c_2 + c_3)k + (c_1 - c_3 + c_4) = ak + b$$

○ 평균 :  $k = n/2$

$$A(n) = a \cdot \frac{n}{2} + b$$

○ 최악 :  $k = n$

$$W(n) = a \cdot n + b$$

(상수  $\times n$  + 상수)



# 순차 탐색의 수행 시간

---

## ○ (상수 $\times n$ + 상수)

- 하나의 상수항은 몇 개의 명령문들의 수행시간을 합쳐 놓은 것
- 컴퓨터의 성능/종류에 종속적이므로 각 명령마다 부여한 상이한 수행시간이 별 의미가 없음
- 간단히 명령마다 동일한 시간 내에 처리된다고 가정해도 무방

# 순차 탐색의 수행 시간

---

- 상수항보다  $n$ 의 차수가 더 중요
  - 상수 값을 무시하고 단지 입력 크기에 대한 차수만 고려
    - 데이터의 개수가 많을 경우 알고리즘의 수행 시간이 보다 중요한 의미를 가짐
  - 이 경우 상수항이 아닌  $n$ 의 최고 차수에 좌우
    - 하위 차수는 무시

# 순차 탐색의 수행 시간

## ○ 5개 알고리즘의 $n$ 값에 따른 수행시간 비교

알고리즘	1	2	3	4	5
시간( $\mu$ s)	$33n$	$46n \lg n$	$13n^2$	$3.4n^3$	$2^n$
$n=10$	.00033초	.0015초	.0013초	.0034초	.001 초 $4 \times 10^4$ 세기
$n=100$	.003초	.03초	.13초	3.4초	
$n=1000$	.033초	.45초	13초	.94시간	
$n=10,000$	.33초	6.1초	22분	39일	
$n=100,000$	3.3초	1.3분	1.5일	108년	
처리가능 최대 입력 개수					
1초	30,000	2,000	280	67	20
1분	1,800,000	82,000	2,200	260	26

알고리즘

# 점근성능

# 점근 성능

- 입력 크기  $n$ 이 커질 때의 알고리즘의 성능
  - 입력의 크기  $n$ 이 커짐에 따라 상수항과 차수가 낮은 항들의 역할이 감소  $\rightarrow$   $n$ 의 최고 차수에 의존

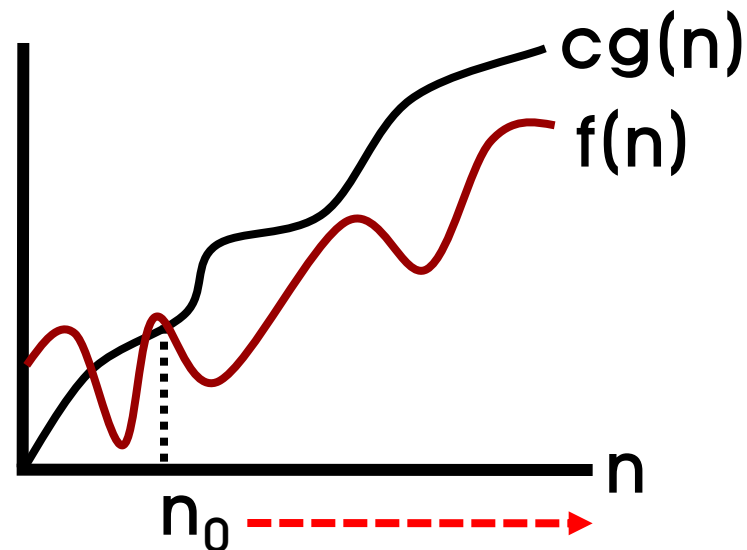
	$10n+9$	$n^2/2+3n$
$n=5$	59	27.5
$n=10$	109	80
$n=15$	159	157.5
$n=16$	169	176
$n=20$	209	260

# 점근 성능의 표기법

## 정의

‘Big-oh’ 점근적 상한

(1)  $n \geq n_0$ 인 모든  $n$ 에 대하여  $f(n) \leq c \cdot g(n)$ 을 만족하는 양의 상수  $c$ ,  $n_0$ 이 존재하면  $f(n) = O(g(n))$ 이다.

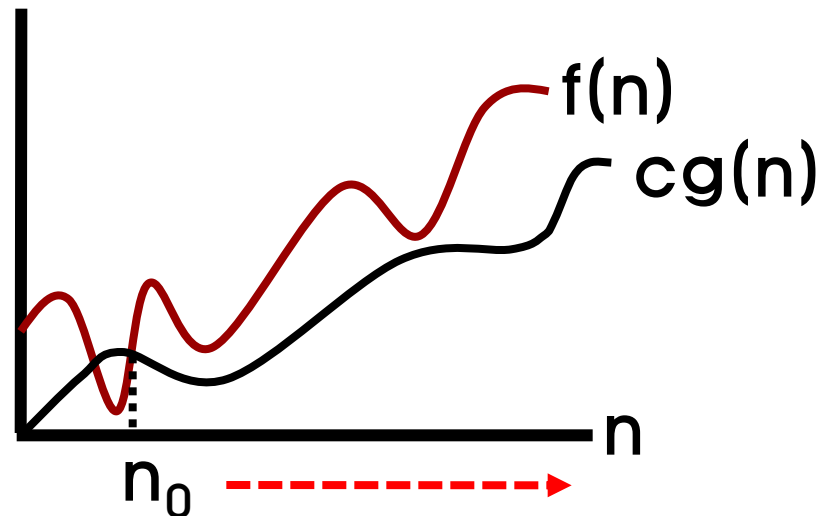


# 점근 성능의 표기법

## 정의

‘Big-omega’ 점근적 하한

(2)  $n > n_0$ 인 모든  $n$ 에 대하여  $f(n) \geq c \cdot g(n)$ 을 만족하는 양의 상수  $c$ ,  $n_0$ 이 존재하면  $f(n) = \Omega(g(n))$ 이다.



# 점근 성능의 표기법

## 정의

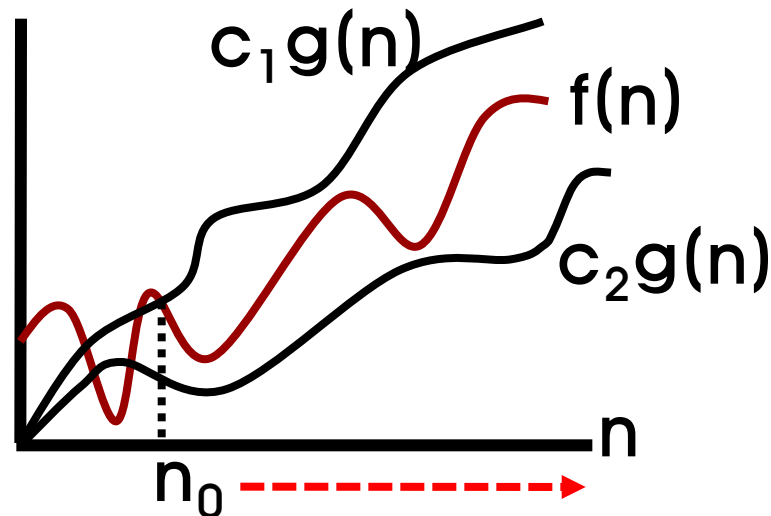
‘Big-theta’ 점근적 상하한

(3)  $n \geq n_0$ 인 모든  $n$ 에 대하여

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ 을 만족하는 양의 상수

$c_1, c_2, n_0$  이 존재하면, 즉

$f(n) = \Omega(g(n)) = O(g(n))$ 이면  $f(n) = \Theta(g(n))$





# 점근 성능의 표기법

- $f(n)=3n+3$ ,  $g(n)=n$ 
  - $(c=5, n_0=2 \text{ 일 때})$   $n \geq 2$ 에 대해서  $3n+3 \leq 5 \cdot n \rightarrow f(n)=O(g(n))=O(n)$
  - $(c=3, n_0=1 \text{ 일 때})$   $n \geq 1$ 에 대해서  $3n+3 \geq 3 \cdot n \rightarrow f(n)=\Omega(g(n))=\Omega(n)$
  - $f(n)=O(n)$  and  $f(n)=\Omega(n) \rightarrow f(n)=\Theta(n)$
- $f(n)=3n^3+3n-1$ ,  $g(n)=n^3$ 
  - $(c=7, n_0=1 \text{ 일 때})$   $n \geq 1$ 에 대해서  $3n^3+3n-1 \leq 7 \cdot n^3 \rightarrow f(n)=O(n^3)$
  - $(c=2, n_0=2 \text{ 일 때})$   $n \geq 3$ 에 대해서  $3n^3+3n-1 \geq 2 \cdot n^3 \rightarrow f(n)=\Omega(n^3)$
  - $f(n)=O(n^3)$  and  $f(n)=\Omega(n^3) \rightarrow f(n)=\Theta(n^3)$

# 점근 성능의 표기법

○  $f(n)=2n^3+3n^2-n+10$ ,  $g(n)=n^3$

- (c=5, n0=5일 때)  $n>5$ 에 대해서

$$2n^3+3n^2-n+10 \leq 5 \cdot n^3 \quad (2 \times 5^3 + 3 \times 5^2 - 5 + 10) \leq (5 \times 5^3)$$

→  $f(n)=O(n^3)$

$$336 = ((2 \times 125) + 75 + 5) \quad 625 = (5 \times 125)$$

- (c=2, n0=3일 때)  $n>3$ 에 대해서

$$2n^3+3n^2-n+10 \geq 2 \cdot n^3 \quad (2 \times 3^3 + 3 \times 3^2 - 5 + 10) \leq (2 \times 3^3)$$

→  $f(n)=\Omega(n^3)$

$$91 = ((2 \times 27) + 27 + 5) \quad 54 = (2 \times 27)$$

- $f(n)=O(n^3)$  and  $f(n)=\Omega(n^3)$  →  $f(n)=\Theta(n^3)$

# 점근 성능의 표기법

- 주요 함수 간의 연산시간의 크기 관계

$$O(1) < O(\log n) < O(n) < O(n \log n) \\ < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

← 효율적

비효율적 →

$$10n+9 \\ \downarrow \\ O(n)$$

$$n^2/2+3n \\ \downarrow \\ O(n^2)$$

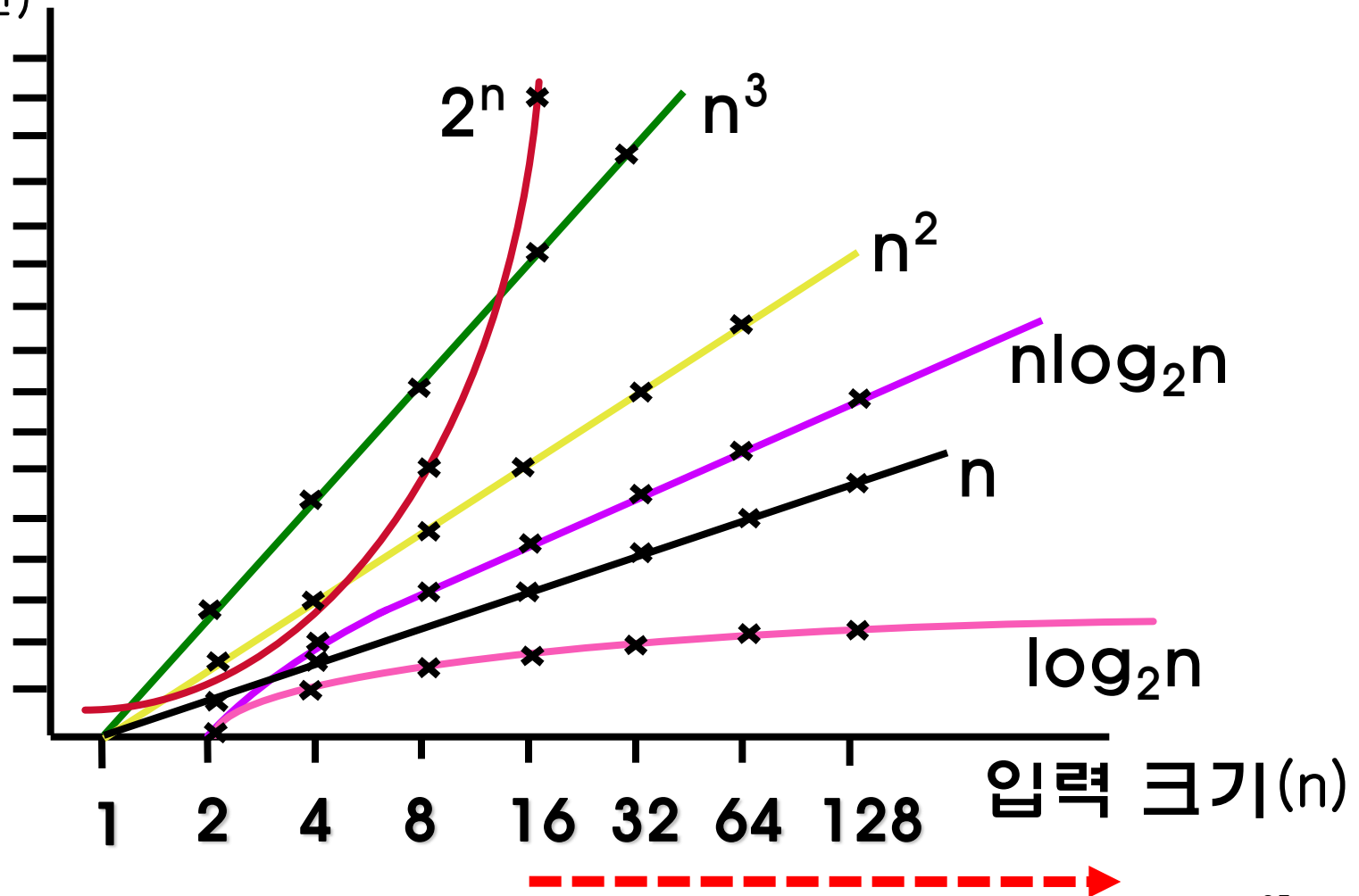
# 입력 크기에 따른 연산의 증가 추세

<b>logn</b>	<b>n</b>	<b>nlogn</b>	<b>n<sup>2</sup></b>	<b>n<sup>3</sup></b>	<b>2<sup>n</sup></b>
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296

# 함수의 증가율

(수행시간)

값



# 시간 복잡도와 표기법

---

- 알고리즘의 시간복잡도를 구하려면
  - 알고리즘 계산시간(연산의 수행 빈도수)  $f(n)$ 을 구한 후  $f(n)=O(g(n))$ 을 만족하는 최소의  $g(n)$ 을 찾음
  - 실용적인 접근 방법
    - 알고리즘에 나타난 루프의 반복회수를 조사
    - ∴  $g(n)$ 은 최고 차수에 지배

# 시간 복잡도와 표기법

```
i = 1;
while (i <= n) {
    x = x + 1;
    i = i + 1;
}
```

$3n + 2 \rightarrow O(n)$

```
int i, j;
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        if (j > i)
            C[i,j] = A[i]*B[j];
```

$O(n^2)$

알고리즘

# 순환 알고리즘의 성능



# 순환 알고리즘의 성능

- 순환 알고리즘의 수행시간을 나타내기 위한 방법으로 점화식이 사용됨

```
BinarySearch (A[], key, left, right)
```

```
{
```

```
1  mid = (left + right)/2;
```

```
2  if (A[mid] = key) return(mid);
```

```
3  else if (A[mid] > key) BinarySearch(A, key, left, mid-1)
```

```
4      else BinarySearch(A, key, mid+ 1, right);
```

```
}
```

$$T(n) = T(n/2) + O(1)$$

# 점화식

$$T(n) = T(n/2) + c_2, \quad T(1) = c_1$$

$$\begin{aligned} T(n) &= T(n/2) + c_2 \\ &= T(n/2^2) + c_2 + c_2 = T(n/2^2) + 2c_2 \\ &= T(n/2^3) + c_2 + c_2 + c_2 = T(n/2^3) + 3c_2 \\ &\dots \\ &= T(n/2^{k-1}) + (k-1)c_2 \\ &= T(n/2^k) + kc_2 = T(1) + kc_2 \\ &= c_1 + kc_2 \\ &= c_1 + c_2 \log_2 n \\ &= \Theta(\log n) \end{aligned}$$

$n = 2^k$ 인 경우에만 정의 가능 ( $k = \log_2 n$ )

# 기본 점화식과 폐쇄형

$$(1) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(1), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n)$$

$$(2) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n-1) + \Theta(n), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n^2)$$

$$(3) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n/2) + \Theta(1), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(\log n)$$

$$(4) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(n/2) + \Theta(n), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n)$$

$$(5) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T(n/2) + \Theta(1), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n)$$

$$(6) \quad T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T(n/2) + \Theta(n), & n \geq 2 \end{cases}$$

$$\Rightarrow T(n) = \Theta(n \log n)$$