

# Data Structure

## Chapter 1. 자료구조와 알고리즘

Dong Kyue Kim

Hanyang University  
dqkim@hanyang.ac.kr

# 자료구조와 알고리즘

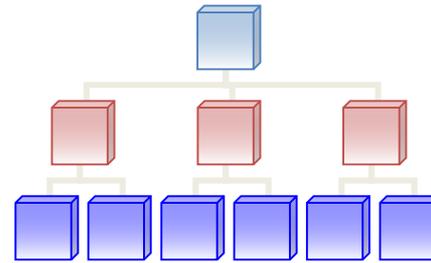
ESLAB

# 일상생활에서의 사물의 조직화

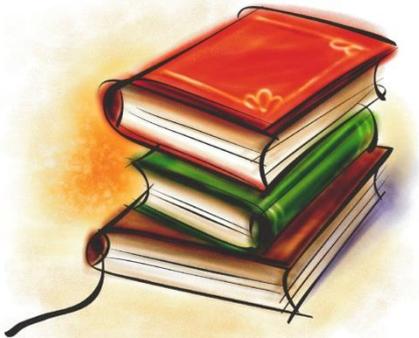
해야할일  
리스트



조직도



일상생활에서의  
사물의 조직화



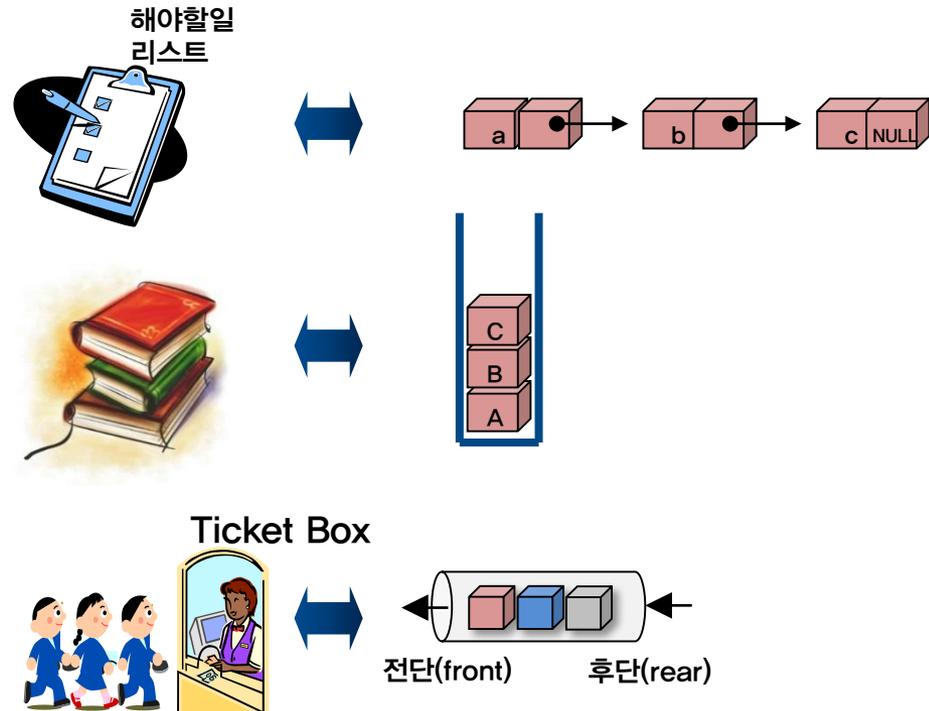
Ticket Box



# 일상생활과 자료구조의 비교

## 일상생활 vs 자료구조

자료구조	일상생활에서의 예
스택	물건을 쌓아두는 것
큐	영화관 매표소의 줄
리스트	할일 리스트
사전, 탐색구조	영어사전
그래프	지도
트리	조직도



# 자료구조와 알고리즘

• 프로그램 = 자료구조 + 알고리즘

– 예) 최대값 탐색 프로그램 = 배열 + 순차탐색

자료구조

알고리즘

score[ ]

80	70	90	...	30
----	----	----	-----	----

```
tmp ← score[0];  
for i ← 1 to n do  
    if score[i] > tmp  
        then tmp ← score[i];
```

# 알고리즘

- 알고리즘(algorithm) : 컴퓨터로 문제를 풀기위한 단계적인 절차
- 알고리즘의 조건
  - 입력 : 0개 이상의 입력이 존재하여야 한다.
  - 출력 : 1개 이상의 출력이 존재하여야 한다.
  - 명백성 : 각 명령어의 의미는 모호하지 않고 명확해야 한다.
  - 유한성 : 한정된 수의 단계 후에는 반드시 종료되어야 한다.
  - 유효성 : 각 명령어들은 실행 가능한 연산이어야 한다.

## 알고리즘의 기술 방법

- 영어나 한국어와 같은 자연어
- 흐름도(Flow Chart)
- 유사 코드(Pseudo Code)
- C와 같은 프로그래밍 언어

# 알고리즘의 기술 방법(자연어)

## • 자연어

- 인간이 읽기가 쉬움
- 자연어의 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해짐
- 예) 배열에서 최대값 찾기 알고리즘

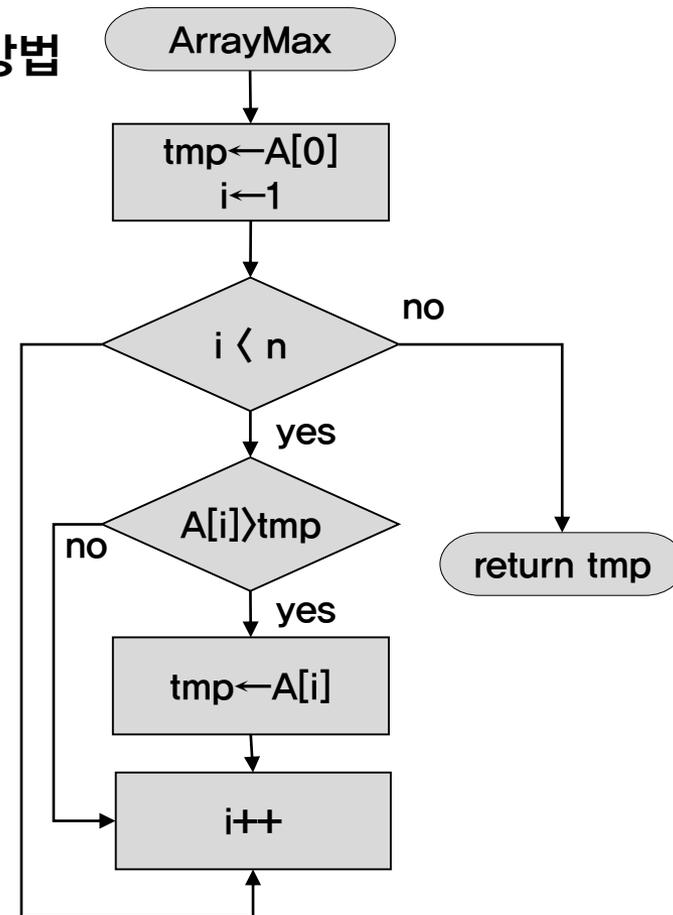
ArrayMax(A,n)

1. 배열 A의 첫번 요소를 변수 tmp에 복사
2. 배열 A의 다음 요소들을 차례대로 tmp와 비교하면 더 크면 tmp로 복사
3. 배열 A의 모든 요소를 비교했으면 tmp를 반환

# 알고리즘의 기술 방법(흐름도)

## • 흐름도(Flow Chart)

- 직관적이고 이해하기 쉬운 알고리즘 기술 방법
- 복잡한 알고리즘의 경우 상당히 복잡해짐
- 예) 배열에서 최대값 찾기 알고리즘



# 알고리즘의 기술 방법(유사코드)

- 유사코드(Pseudo Code)
  - 알고리즘의 고수준 기술 방법
  - 자연어보다는 더 구조적인 표현 방법
  - 프로그래밍 언어보다는 덜 구체적인 표현방법
  - 알고리즘 기술에 가장 많이 사용
  - 프로그램을 구현할 때의 여러 가지 문제들을 감출 수 있다.
  - 알고리즘의 핵심적인 내용에만 집중

```
ArrayMax(A,n)
```

```
tmp ← A[0];  
for i←1 to n-1 do  
    if tmp < A[i] then  
        tmp ← A[i];  
return tmp;
```

# 알고리즘의 기술 방법(프로그래밍 언어)

- 프로그래밍 언어

- 알고리즘의 가장 정확한 기술이 가능
- 실제 구현 시, 많은 구체적인 사항들이 알고리즘의 핵심적인 내용에 대한 이해를 방해

```
#define MAX_ELEMENTS 100
int score[MAX_ELEMENTS];
int find_max_score(int n)
{
    int i, tmp;
    tmp=score[0];
    for( i = 1; i < n; i++){
        if( score[i] > tmp ){
            tmp = score[i];
        }
    }
    return tmp;
}
```

# 버블정렬 (Bubble Sort)

- 인접한 2개의 레코드를 비교하여 순서대로 되어 있지않으면 서로 교환
- 이러한 비교-교환 과정을 리스트의 왼쪽 끝에서 오른쪽 끝까지 반복(스캔)



# 버블정렬 알고리즘

```
#define SWAP(x, y, t) ( (t)=(x), (x)=(y), (y)=(t) )  
void bubble_sort(int list[], int n)  
{ int i, j, temp;  
  for( i = n-1; i > 0; i-- )  
    for( j = 0; j < i; j++ )  
      if( list[j] > list[j+1] )  
        SWAP(list[j], list[j+1], temp);  
}
```

# 추상 데이터 타입

ESLAB

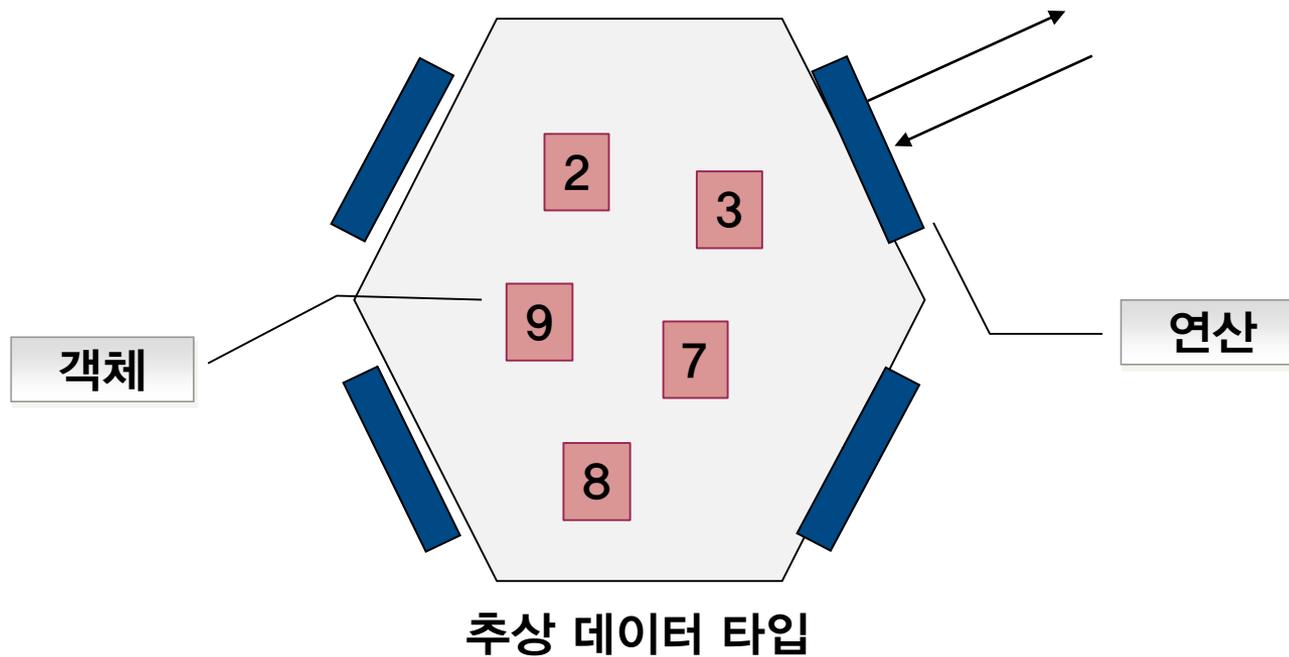
# 데이터 타입

- 데이터 타입(Data Type)
  - 데이터의 집합과 연산의 집합
  - 예) int 데이터 타입
    - 데이터 : { $\dots$ , -2, -1, 0, 1, 2,  $\dots$ }
    - 연산 : +, -, /, \*, %
- 추상 데이터 타입(ADT: Abstract Data Type)
  - 데이터 타입을 추상적(수학적)으로 정의한 것
  - 데이터나 연산이 **무엇(what)**인가는 정의되지만 데이터나 연산을 **어떻게(how)** 컴퓨터 상에서 구현할 것인지는 정의되지 않는다.

# 추상 데이터 타입의 정의

## • 추상 데이터 타입

- 객체: 추상 데이터 타입에 속하는 객체가 정의된다.
- 연산: 이들 객체들 사이의 연산이 정의된다. 이 연산은 추상 데이터 타입과 외부를 연결하는 인터페이스의 역할을 한다.



## 추상 데이터 타입의 예시

- Nat\_No

- 객체: 0에서 시작하여 INT\_MAX까지의 순서화된 정수의 부분범위
- 연산

```
zero()      ::= return 0;
is_zero()   ::= if (x) return FALSE;
              else return TRUE;
add(x,y)    ::= if( (x+y) <= INT_MAX ) return x+y;
              else return INT_MAX
sub(x,y)    ::= if ( x<y ) return 0;
              else return x-y;
equal(x,y)  ::= if( x=y ) return TRUE;
              else return FALSE;
successor(x) ::= if( (x+y) <= INT_MAX )
                 return x+1;
```

## 추상 데이터 vs VCR

### • 추상 데이터

- 제공하는 연산만을 사용가능
- 어떻게 사용하는지를 알아야함
- 추상 데이터 타입 내부의 데이터 접근불가
- 어떻게 구현되었는지 몰라도 사용가능
- 구현이 변경되어도 인터페이스가 변경되지 않으면 그대로 사용가능

### • VCR

- 인터페이스가 제공하는 특정한 작업만을 수행
- 사용자는 이러한 작업들에 대한 이해 필요  
(비디오를 시청하기 위해서는 무엇을 해야 하는지를 알아야함)
- VCR의 내부를 볼 수 없음
- VCR의 내부에서 무엇이 일어나고 있는지 몰라도 사용가능
- 내부의 기계장치를 교환되어도 인터페이스가 바뀌지 않으면 그대로 사용가능

# 알고리즘의 성능 분석

EMBEDDED SECURITY SYSTEM LABORATORY

# 알고리즘의 성능 분석

## • 알고리즘의 성능 분석 기법

### – 수행 시간 측정

- 두개의 알고리즘의 실제 수행 시간을 측정하는 것
- 실제로 구현하는 것이 필요
- 동일한 하드웨어를 사용하여야 함

### – 알고리즘의 복잡도 분석

- 직접 구현하지 않고서도 수행 시간을 분석하는 것
- 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
- 일반적으로 연산의 횟수는  $n$ 의 함수
- **시간 복잡도 분석**: 수행 시간 분석
- **공간 복잡도 분석**: 수행시 필요로 하는 메모리 공간 분석

# 알고리즘의 성능 분석(수행 시간)

## • 수행 시간 측정

- 컴퓨터에서 수행시간을 측정하는 방법에는 주로 clock 함수가 사용된다.
- clock\_t clock(void);
  - clock 함수는 호출되었을 때의 시스템 시각을 CLOCKS\_PER\_SEC 단위로 반환
- 수행시간을 측정하는 전형적인 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void main( void )
{
    clock_t start, finish;
    double duration;
    start = clock();
    // 수행시간을 측정하고 하는 코드....
    // ....
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("%f 초입니다.\n", duration);
}
```

# 알고리즘의 성능 분석(복잡도 분석)

## • 시간 복잡도

- 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표시
- 산술 연산, 대입 연산, 비교 연산, 이동 연산의 기본적인 연산
- 수행시간이 입력의 크기에 따라 변하면 안됨
- 알고리즘이 수행하는 연산의 개수를 계산하여 두 개의 알고리즘 비교가능
- 시간 복잡도 함수 : 입력의 개수  $n$ 에 대한 연산의 수행횟수.  $T(n)$  이라고 표기

# 복잡도 분석의 예

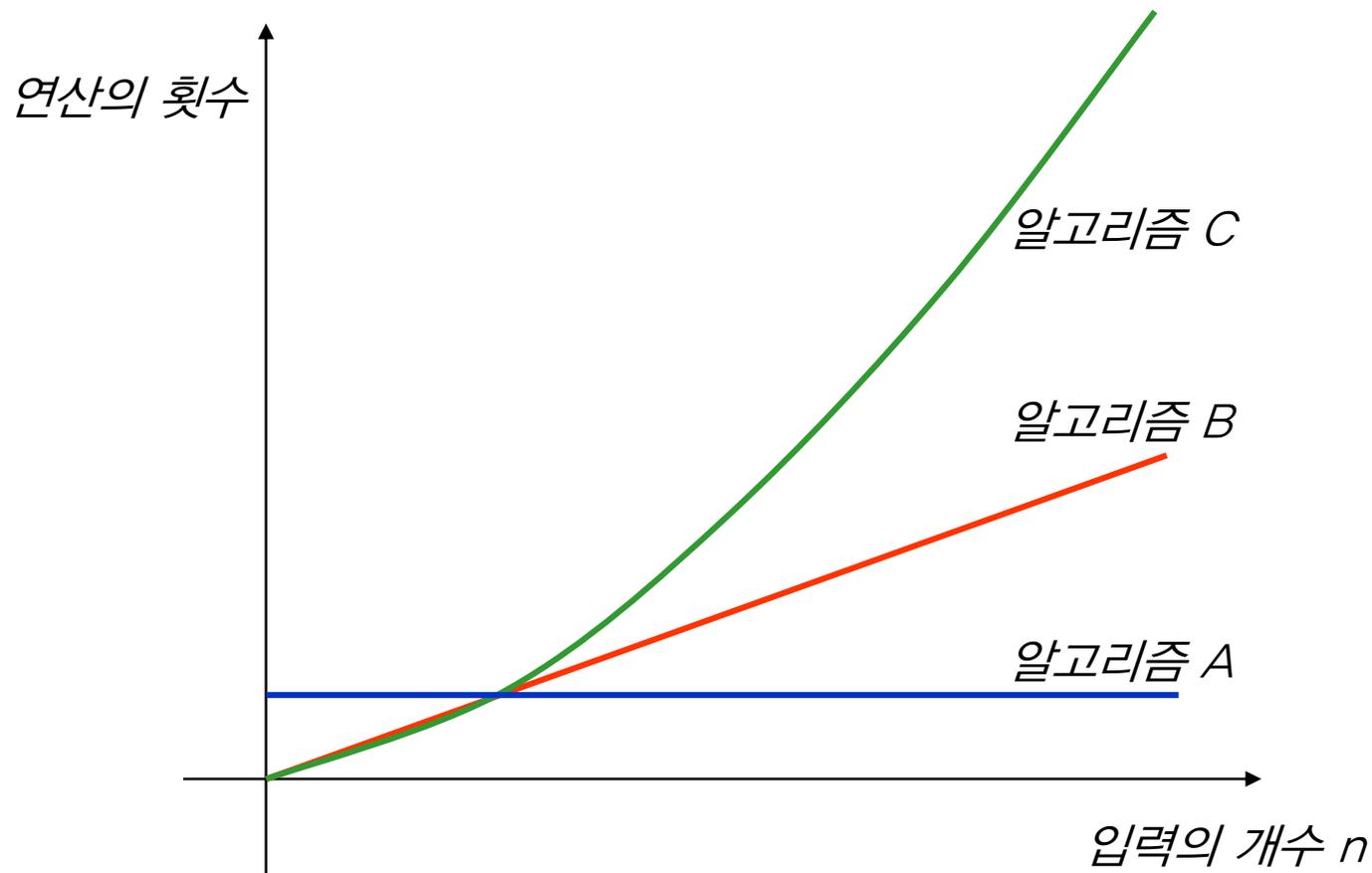
- $n$ 을  $n$ 번 더하는 문제
  - 각 알고리즘이 수행하는 연산의 개수를 셈
  - 단, for 루프 제어 연산은 고려X

알고리즘 A	알고리즘 B	알고리즘 C
$\text{sum} \leftarrow n * n;$	$\text{sum} \leftarrow 0;$ <b>for</b> $i \leftarrow 1$ to $n$ <b>do</b> $\text{sum} \leftarrow \text{sum} + n;$	$\text{sum} \leftarrow 0;$ <b>for</b> $i \leftarrow 1$ to $n$ <b>do</b> <b>for</b> $j \leftarrow 1$ to $n$ <b>do</b> $\text{sum} \leftarrow \text{sum} + 1;$

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	$n + 1$	$n * n + 1$
덧셈연산		$n$	$n * n$
곱셈연산	1		
나눗셈연산			
전체연산수	2	$2n + 1$	$2n^2 + 1$

# 연산의 횟수를 그래프로 표현

## • 그래프 표현



## 시간 복잡도 함수 계산 예

- 코드를 분석해보면 수행되는 연산들의 횟수를 입력 크기의 함수로 만들 수 있다.

ArrayMax(A,n)

```
tmp ← A[0];  
for i←1 to n-1 do  
    if tmp < A[i] then  
        tmp ← A[i];  
return tmp;
```

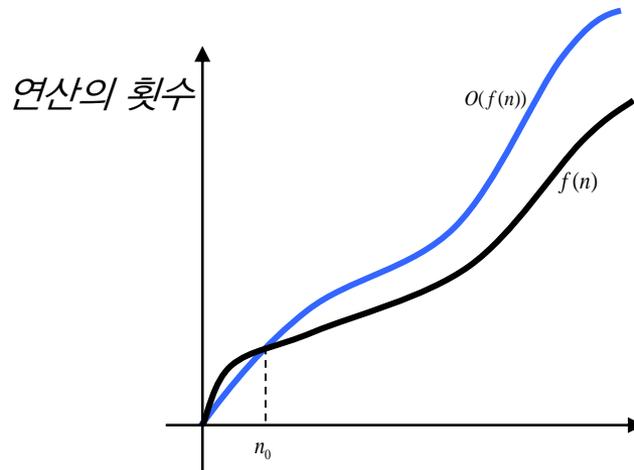
1번의 대입 연산  
루프 제어 연산은 제외  
n-1번의 비교 연산  
n-1번의 대입 연산(최대)  
1번의 반환 연산

총 연산수 =  $2n$ (최대)

# 빅오 표기법

## • 빅오 표기법

- 연산의 횟수를 대략적(점근적)으로 표기한 것
- 두개의 함수  $f(n)$ 과  $g(n)$ 이 주어졌을 때,  
모든  $n \geq n_0$ 에 대하여  $|f(n)| \leq c|g(n)|$ 을 만족하는 2개의 상수  $c$ 와  $n_0$ 가 존재하면  $f(n) = O(g(n))$ 이다.
- 빅오는 **함수의 상한**을 표시한다.
  - 예)  $n \geq 5$  이면  $2n+1 < 10n$  이므로  $2n+1 = O(n)$



## 빅오 표기법의 예

### • 예제 1.1 빅오 표기법

-  $f(n) = 5$ 이면  $O(1)$ 이다.

$n_0 = 1, c = 10$ 일 때,  $n \geq 1$ 에 대하여  $5 \leq 10 \cdot 1$ 이 된다.

-  $f(n) = 2n + 1$ 이면  $O(n)$ 이다.

$n_0 = 2, c = 3$ 일 때,  $n \geq 2$ 에 대하여  $2n + 1 \leq 3n$ 이 된다.

-  $f(n) = 3n^2 + 100$ 이면  $O(n^2)$ 이다.

$n_0 = 100, c = 5$ 일 때,  $n \geq 100$ 에 대하여  $3n^2 + 100 \leq 5n^2$ 이 된다.

-  $f(n) = 5 \cdot 2^n + 10n^2 + 100$ 이면  $O(2^n)$ 이다.

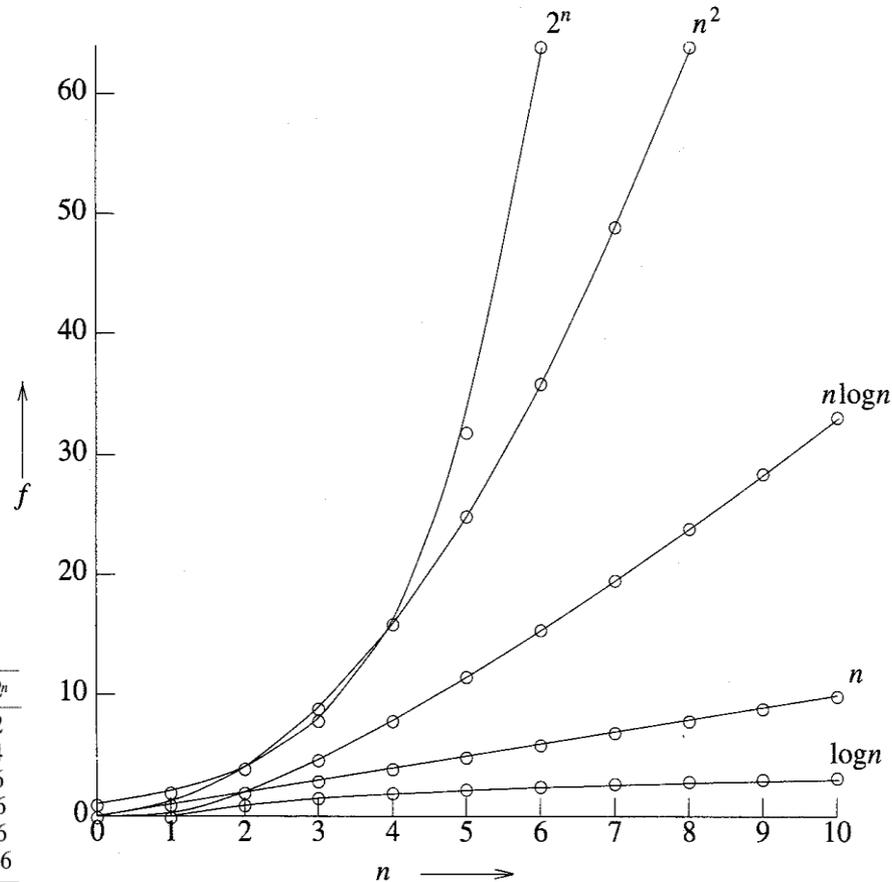
$n_0 = 1000, c = 10$ 일 때,  $n \geq 1000$ 에 대하여  $5 \cdot 2^n + 10n^2 + 100 \leq 10 \cdot 2^n$ 이 된다.

# 빅오 표기법의 종류

## 종류

- $O(1)$  : 상수형
- $O(\log n)$  : 로그형
- $O(n)$  : 선형
- $O(n \log n)$  : 로그선형
- $O(n^2)$  : 2차형
- $O(n^3)$  : 3차형
- $O(n^k)$  : k차형
- $O(2^n)$  : 지수형
- $O(n!)$  : 팩토리얼형

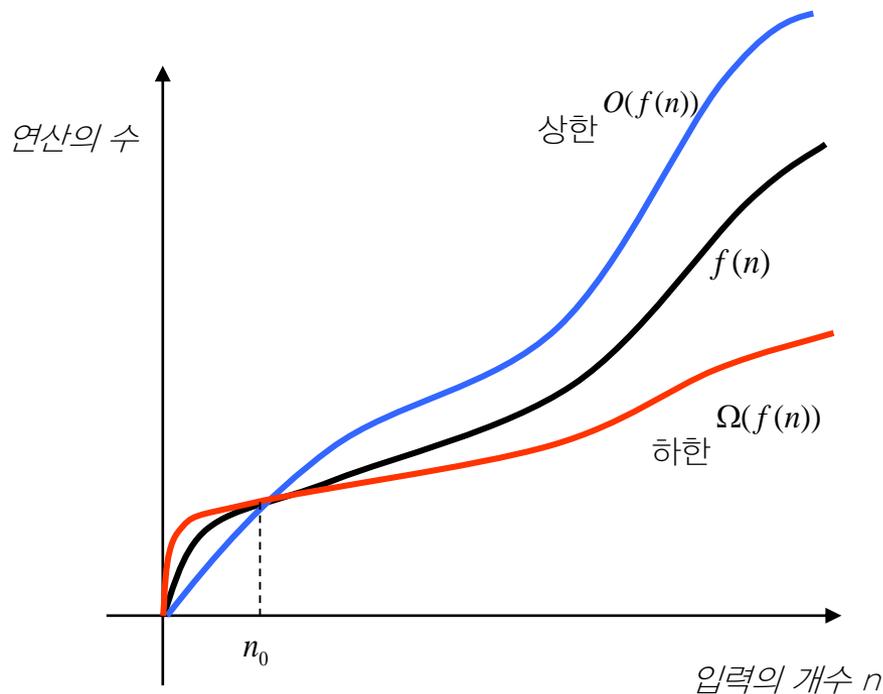
$\log n$	$n$	$n \log n$	$n^2$	$n^3$	$2^n$
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65,536
5	32	160	1064	32,768	4,294,967,296



# 그 외 표기법

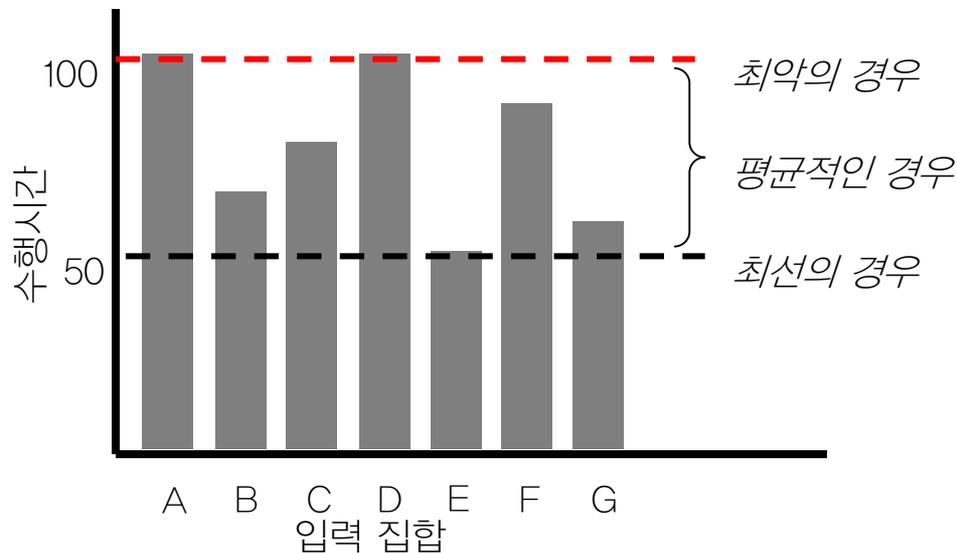
## • 빅오메가 표기법

- 모든  $n \geq n_0$ 에 대하여  $|f(n)| \geq c|g(n)|$ 을 만족하는 2개의 상수  $c$ 와  $n_0$ 가 존재하면  $f(n) = \Omega(g(n))$ 이다.
- 빅오메가는 **함수의 하한**을 표시한다.
  - 예)  $n \geq 5$  이면  $2n+1 < 10n$  이므로  $n = \Omega(n)$



## 최선, 평균, 최악의 경우 (1/3)

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있다.
  - 예) 정렬 알고리즘의 수행 시간
- 최선의 경우(best case) : 수행시간이 가장 빠른 경우
- 평균의 경우(average case) : 수행시간이 평균적인 경우
- 최악의 경우(worst case) : 수행시간이 가장 늦은 경우



# 최선, 평균, 최악의 경우 (3/3)

## • 순차탐색

– 최선의 경우: 찾고자 하는 숫자가 맨앞에 있는 경우

$$\therefore O(1)$$

– 최악의 경우: 찾고자 하는 숫자가 맨뒤에 있는 경우

$$\therefore O(n)$$

– 평균적인 경우: 각 요소들이 균일하게 탐색된다고 가정하면

$$(1+2+\dots+n)/n=(n+1)/2$$

$$\therefore O(n)$$



# 자료 구조 표기법

ESLAB

## 자료 구조의 C언어 표현방법

- 자료구조와 관련된 데이터들을 구조체로 정의
- 연산을 호출할 경우, 이 구조체를 함수의 파라미터로 전달
- 예)

```
// 자료구조 스택과 관련된 자료들을 정의
typedef int element;
typedef struct {
    int top;
    element stack[MAX_STACK_SIZE];
} StackType;

// 자료구조 스택과 관련된 연산들을 정의
void push(StackType *s, element item)
{
    if( s->top >= (MAX_STACK_SIZE -1)){
        stack_full();
        return;
    }
    s->stack[++(s->top)] = item;
}
```

자료구조의 요소

관련된 데이터를 구조체로 정의

연산을 호출할때 구조체를 함수의 파라미터로 전달

## 자료구조 기술규칙 (1/2)

### • 상수

- 대문자로 표기
- 예) #define MAX\_ELEMENT 100

### • 변수의 이름

- 소문자를 사용하였으며 언더라인을 사용하여 단어와 단어를 분리
- 예) int increment;  
int new\_node;

### • 함수의 이름

- 동사를 이용하여 함수가 하는 작업을 표기
- 예) int add(ListNode \*node) // 혼동이 없는 경우  
int list\_add(ListNode \*node) // 혼동이 생길 우려가 있는 경우

## 자료구조 기술규칙 (2/2)

- typedef의 사용
- C언어에서 사용자 정의 데이터 타입을 만드는 경우에 쓰이는 키워드

```
typedef <새로운 타입의 정의> <새로운 타입 이름>;
```

```
- (예) typedef int element;  
      typedef struct ListNode {  
          element data;  
          struct ListNode *link;  
      } ListNode;
```