자료구조 소개 Part 2

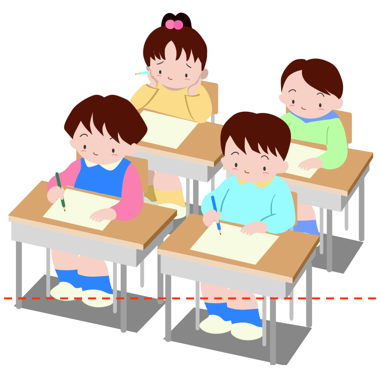
자료구조와 알고리즘

- 프로그램 = 자료구조 + 알고리즘
  - 최대값 탐색 프로그램 = 배열+ 순차탐색

자료구조

score[]

80	70	90	...	30



알고리즘

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

알고리즘 조건

- 알고리즘으로서 성립되기 위한 5가지 조건

조건		설명
1	입력(input)	외부에서 제공되는 데이터가 0개 이상 있다.
2	출력(output)	적어도 한 가지 이상의 결과를 생성한다.
3	명확성(definiteness)	알고리즘을 구성하는 각 명령어들은 그 의미가 명백하고 모호하지 않아야 한다.
4	유한성(infiniteness)	알고리즘의 명령대로 순차적인 실행을 하면 언젠가는 반드시 실행이 종료되어야 한다.
5	유효성(effectiveness)	원칙적으로 모든 명령들은 오류가 없이 실행 가능해야 한다.

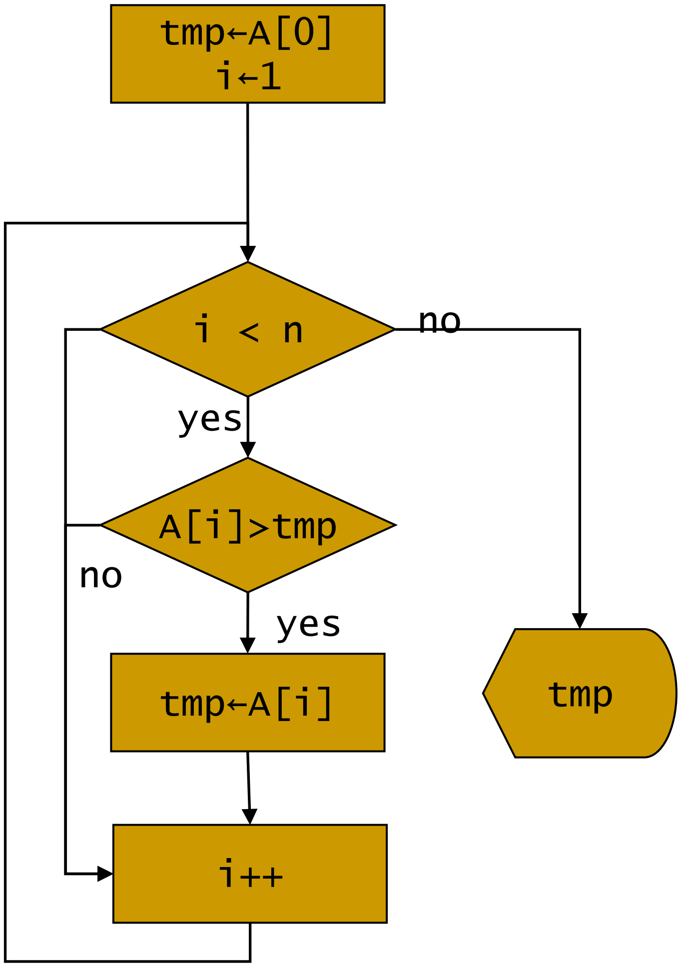
알고리즘 표현 방식

- 영어나 한국어와 같은 **자연어**
  - 인간이 읽기가 쉽다
  - 그러나 자연어의 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다
  - 배열에서 최대값 찾기 알고리즘

ArrayMax(A,n)

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

- **흐름도**(flow chart)
  - 직관적이고 이해하기 쉬운 알고리즘 기술 방법
  - 그러나 복잡한 알고리즘의 경우, 상당히 복잡해짐



tmp←A[0]
i←1
i < n
no
yes
A[i]>tmp
no
yes
tmp←A[i]
tmp
i++

유사 코드(pseudo-code)

- 알고리즘의 고수준 기술 방법으로 알고리즘 기술에 가장 많이 사용
- 자연어보다는 더 구조적인 표현 방법이며, 프로그래밍 언어보다는 덜 구체적인 표현 방법임
- 프로그램을 구현할 때의 여러가지 문제들을 감출 수 있다. 즉 알고리즘의 핵심적인 내용에만 집중할 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;
```

대입 연산자가 ←
임을 유의

추상데이터형 (Abstract Data Type)

- Data
- Data type
  - 협의의 정의 : 자료의 유형 --- int, char, float, double, struct, union, ...
  - 광의의 정의 : 자료와 그 자료 상에서 가능한 연산의 모임
    - 예) integer --- 정수들의 모임, +, -, x, /, ... etc
- Abstract Data Type (ADT) : 추상 자료형
  - 불필요한 데이터나 연산을 없애고 꼭 필요한 항목들로만 구성한 data type으로 데이터와 연산에 대한 명세를 구현으로부터 분리하여 구성
  - 연산의 의미(what)는 정의되어 있지만, 구체적인 연산의 구현(how)이 정의되어 있지 않아 구현자에 의해 다르게 구현될 수 있다

ADT와 실세계 : VCR과 비교

- 사용자들은 추상 데이터 타입이 제공하는 연산만을 사용할 수 있다.
- 사용자들은 추상 데이터 타입을 사용하는 방법을 알아야 한다.
- 사용자들은 추상 데이터 타입 내부의 데이터를 접근할 수 없다.
- 사용자들은 어떻게 구현되었는지 몰라도 이용할 수 있다.
- 만약 다른 사람이 추상 데이터 타입의 구현을 변경하더라도 인터페이스가 변경되지 않으면 사용할 수 있다.

- VCR의 인터페이스가 제공하는 특정한 작업만을 할 수 있다.
- 사용자는 이러한 작업들을 이해해야 한다. 즉 비디오를 시청하기 위해서는 무엇을 해야 하는지를 알아야 한다.
- VCR의 내부를 볼 수는 없다.
- VCR의 내부에서 무엇이 일어나고 있는지를 몰라도 이용할 수 있다.
- 누군가가 VCR의 내부의 기계장치를 교환한다고 하더라도 인터페이스만 바뀌지 않는 한 그대로 사용이 가능하다.

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

복잡도 분석의 예

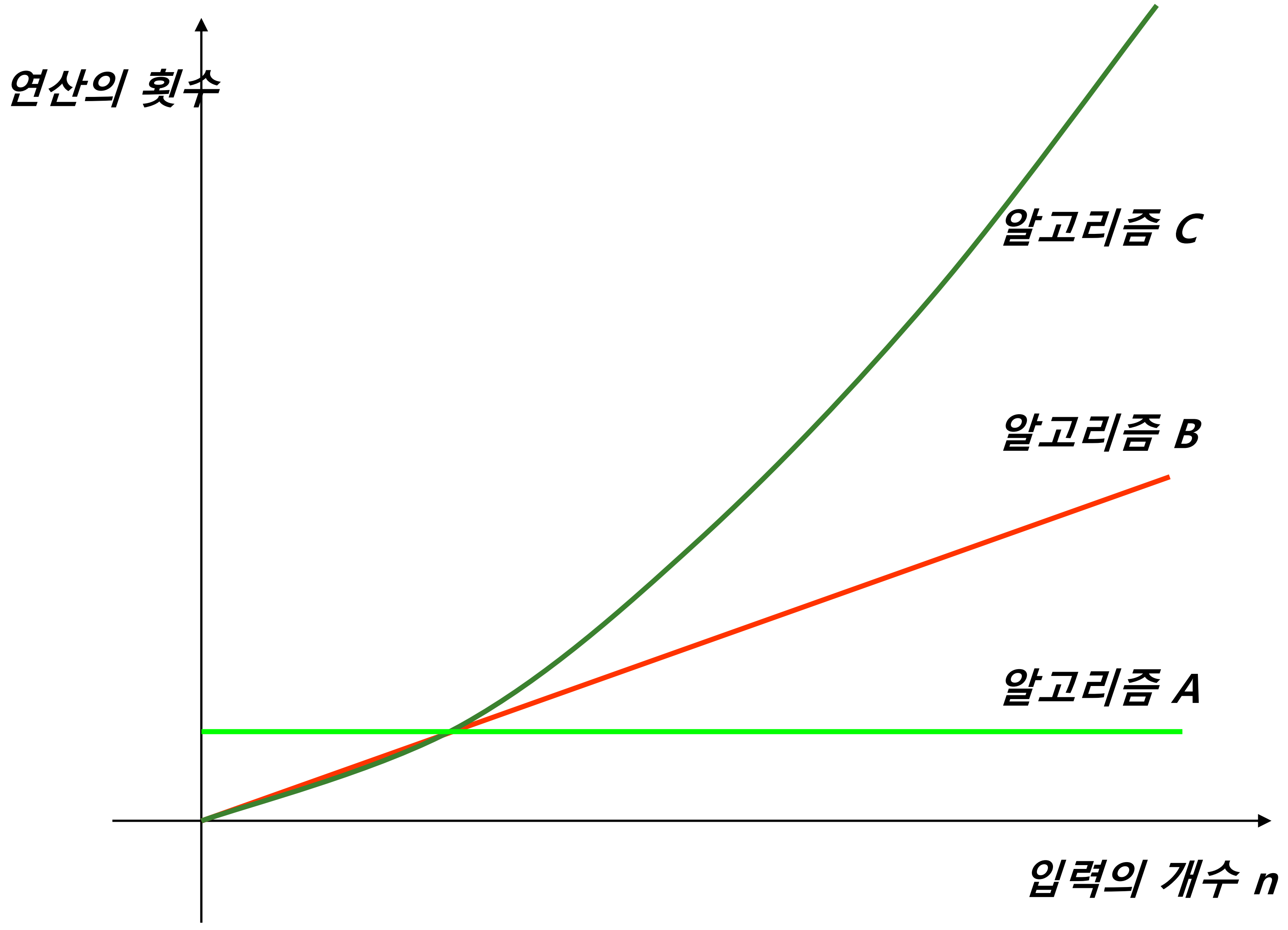
- n을 n번 더하는 문제를 생각해 보자

알고리즘 A	알고리즘 B	알고리즘 C

sum ←n*n;	sum ← 0;
	for i ← 1 to n **do**
	   sum ←sum + n;
		sum ← 0;
		for i←1 to n **do**
		  for ←1 to n **do**
		    sum ←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	$2n^2 + 1$

복잡도를 그래프로 표현



Big-O 표기법

- 자료의 개수가 많은 경우에는 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있다.
  - (예) n=1,000 일 때, $T(n)$의 값은 1,001,001이고 이중에서 첫 번째 항인 n^2 의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지한다.

n=1000인 경우

$T(n)= n^2 + n + 1$

99%

1%

- **Big-O 의 정의**
- 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 c와 n_0가 존재하면 이때, **f(n)=O(g(n))**이라고 한다
- O(f(n))은 함수의 상한을 표시한다.
  - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$

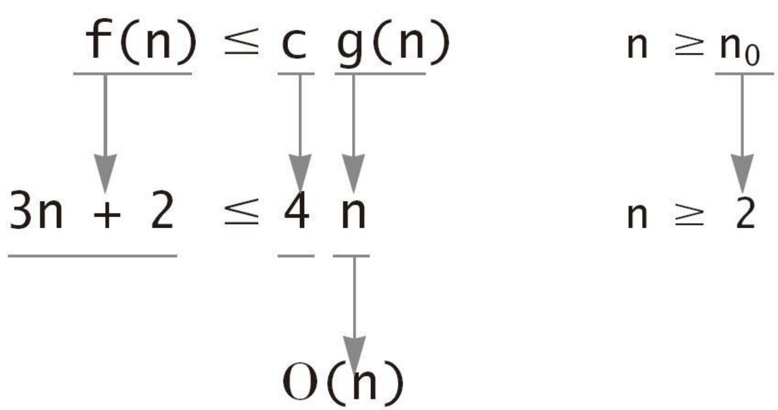
- 빅오 표기법(O)

f(n) = 3n + 2의 Big-O 표기법은 O(n) 이다.

3n + 2 = O (n) 임을 증명하기 위해서는 알고리즘의 효율을

나타내는 O ( )의 괄호 안에 들어가는 함수 g(n) = n일 때

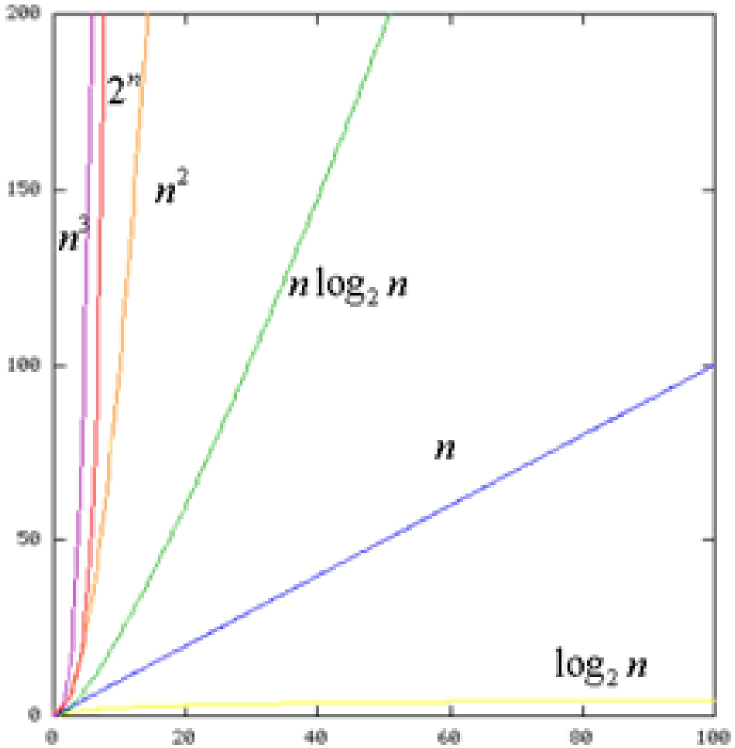
$f(n) \leq cg(n)$인 조건을 만족하는 n_0과 c가 존재함을 보이면 된다.



성능 분석

다음의 Big-O 표기법은 가장 많이 사용되는 것으로서 실행시간이 빠른 순서대로 기술한 것이다.

빅오 표기법	명칭	실행시간
O(1)	상수	위로 갈수록 실행시간이 빠르고 효율적이다.
O(log n)	로그형	
O(n)	선형	
O(nlog n)	선형로그형	
$O(n^2)$	2차형	
$O(n^3)$	3차형	
$O(2^n)$	지수형	
O(n!)	팩토리얼형	



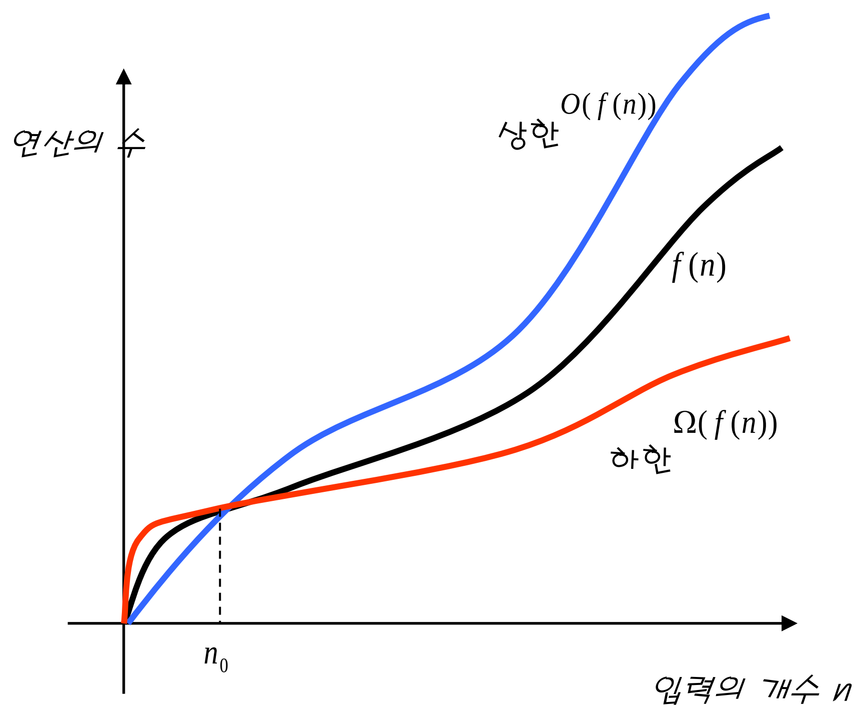
시간복잡도	*n*					
	1	**2**	**4**	**8**	**16**	**32**
1	1	1	1	1	1	1
$\log n$	0	1	2	3	4	5
n	1	2	4	8	16	32
$n \log n$	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
$n!$	1	2	24	40326	20922789888000	26313×10^{33}

Big-Omega : Ω

- Big Omega, Ω 표기법은 Big-O 기호의 반대 개념이다. 알고리즘 수행 시간의 **하한**(Lower Bound)으로서 "최소한 이만한 시간은 걸린다"라는 의미이다.
- **수학적 정의 [오메가(Omega)] :** 모든 n, $n \geq n_0$, 에 대해 $f(n) \geq cg(n)$을 만족하는 두 양의 상수 c와 n_0가 존재하기만 하면 $f(n) = \Omega(g(n))$ (f of n은 "omega" of g of n이라 읽음)이다.

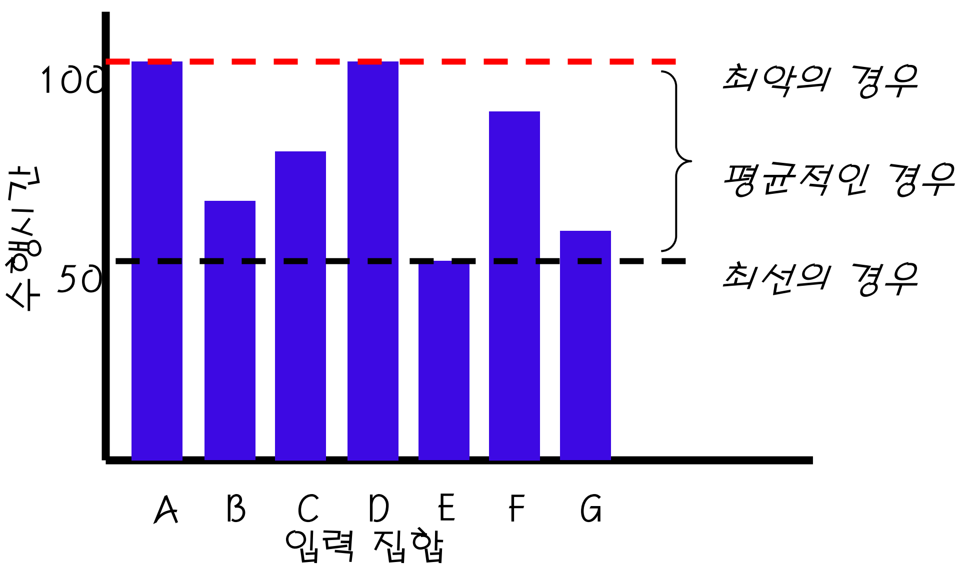
Big-Theta, θ

- 모든 $n \geq n_0$에 대하여 $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$을 만족하는 3개의 상수 c_1, c_2와 n_0가 존재하면 $f(n)=\theta(g(n))$이다.
- Big-theta는 함수의 하한인 동시에 상한을 표시한다
- $f(n)=O(g(n))$이면서 $f(n)= \Omega(g(n))$이면 $f(n)= \theta(n)$이다.
- (예) $n \geq 1$이면 $n \leq 2n+1 \leq 3n$이므로 $2n+1 = \theta(n)$



Best, Average and Worst Case

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



- Best case: 의미가 없는 경우가 많다.
- Average: 계산하기가 상당히 어려움.
- Worst: 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.

순차 탐색의 예

- 최선의 경우:
  - 찾고자 하는 숫자가 맨 앞에 있는 경우 : O(1)
- 최악의 경우:
  - 찾고자 하는 숫자가 맨 뒤에 있는 경우 : O(n)
- 평균적인 경우:
  - 각 요소들이 균일하게 탐색된다고 가정하면 : O(n)
  - $(1+2+3+\ldots+n)/n = (n+1)/2$