Chap. 1
Data Structure & Algorithms

SANGJI University
Kwangman Ko
(kkman@sangji.ac.kr)

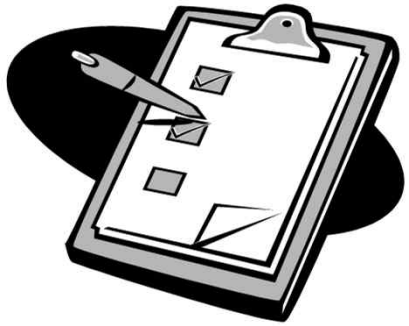
자료와 정보

- 자료(data)
  - 현실 세계로부터의 단순한 관찰이나 측정을 통하여 수집한 사실이나 개념의 값들 또는 값들의 집합.

- 정보(information)
  - 의사 결정에 도움을 주기 위해 유용한 형태로 다시 작성된(=가공) 자료.

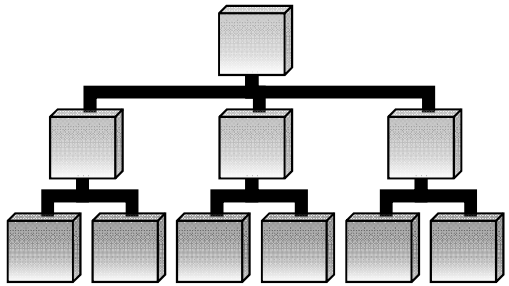
자료구조(data structure) 정의 ?

- 일상 생활에서 사물의 조직화

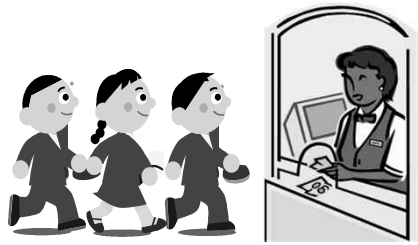


해야할일
리스트

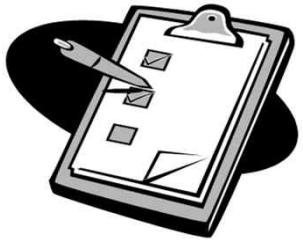
조직도

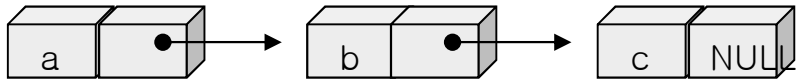


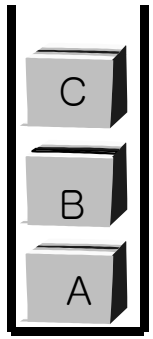
Ticket Box



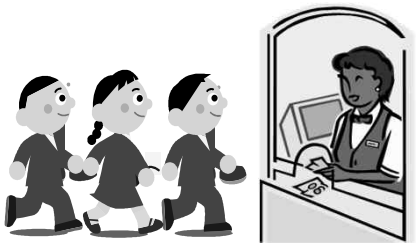
해야할일
리스트

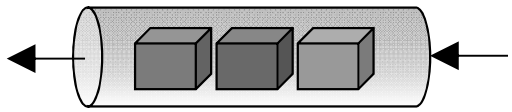






Ticket Box





전단(front)　　후단(rear)

- 자료 구조(data structure) 란 ?
  - 다루고자 하는 자료 원소들(elements) 간 논리적 관계를 기술
  - 컴퓨터의 휘발성 또는 비 휘발성 메모리에 존재하는 자료의 집합
  - 자료 값에 대한 연산을 효율적으로 처리할 수 있도록 자료의 구성을 조직적이고 체계적으로 표현하는 것

자료구조의 선택 기준

- 자료의 양(자료가 차지하는 메모리 공간)
- 자료의 활용 빈도(저장 방식 결정)
- 사용 가능한 기억 용량(전체적인 메모리 공간)
- 자료의 갱신 정도
- 처리 시간의 제한성
- 프로그래밍의 용이성

자료구조의 목적

- 효율성(Efficiency)
  - 좀 더 효율적인 알고리즘이 될 수 있도록 자료를 구조화.
- 추상화(Abstraction)
  - 자료를 보다 쉽게 이해할 수 있는 방법인 추상화를 제공.
- 재사용성(Reusability)
  - 모듈화되어 있고 문맥에 자유롭기(context-free) 때문에 재사용이 가능.

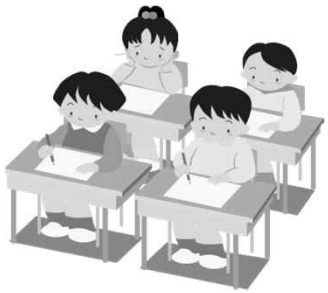
알고리즘과 프로그램

- 프로그램 = 자료구조 + 알고리즘
  - 최대값 탐색 프로그램 예
    - 값을 저장하는 구조, 자료구조 = 배열
    - 최대값을 탐색하는 방법, 알고리즘 = 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(algorithm)
  - 어떤 문제를 해결하기 위해 기술해 놓은 명확한 절차로, 일련의 명령(instruction 또는 step) 집합을 의미

- 프로그램(program)
  - 컴퓨터가 수행할 수 있는 상세화된 명령어의 집합

알고리즘의 요건

- 입력(Input)
  - 제공되는 자료가 있을 수도 있고 없을 수도 있다.
- 출력(Output)
  - 한 개 이상의 결과가 반드시 생성되어야 한다.
- 명백성(Definiteness)
  - 문제 해결 단계에서 수행할 내용은 모호하지 않고 명백해야 한다.
- 유한성(Finiteness)
  - 한정된 수의 단계 후에는 반드시 종료되어야 한다.
- 유효성(Effectiveness)
  - 알고리즘의 모든 명령은 수행 가능한 것이어야 한다.

대표적인 알고리즘

- 삽입(insert)
  - 자료구조에서 새로운 자료를 삽입
- 검색/탐색(search)
  - 자료구조에서 원하는 자료를 찾기
- 삭제/제거(delete)
  - 자료구조에서 기존의 자료를 삭제
- 정렬(sorting)
  - 자료구조에 있는 자료들을 특정 키 값에 의해서 순서대로 나열
- 반복, 순회, …

알고리즘 기술 방법

- 영어나 한국어와 같은 자연어
- 흐름도(flow chart)
- 유사 코드(pseudo-code)
- C/C++/Java/…와 같은 프로그래밍 언어

자연어로 표기된 알고리즘

- 특징
  - 인간이 읽기가 쉽다.
  - 자연어의 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다.

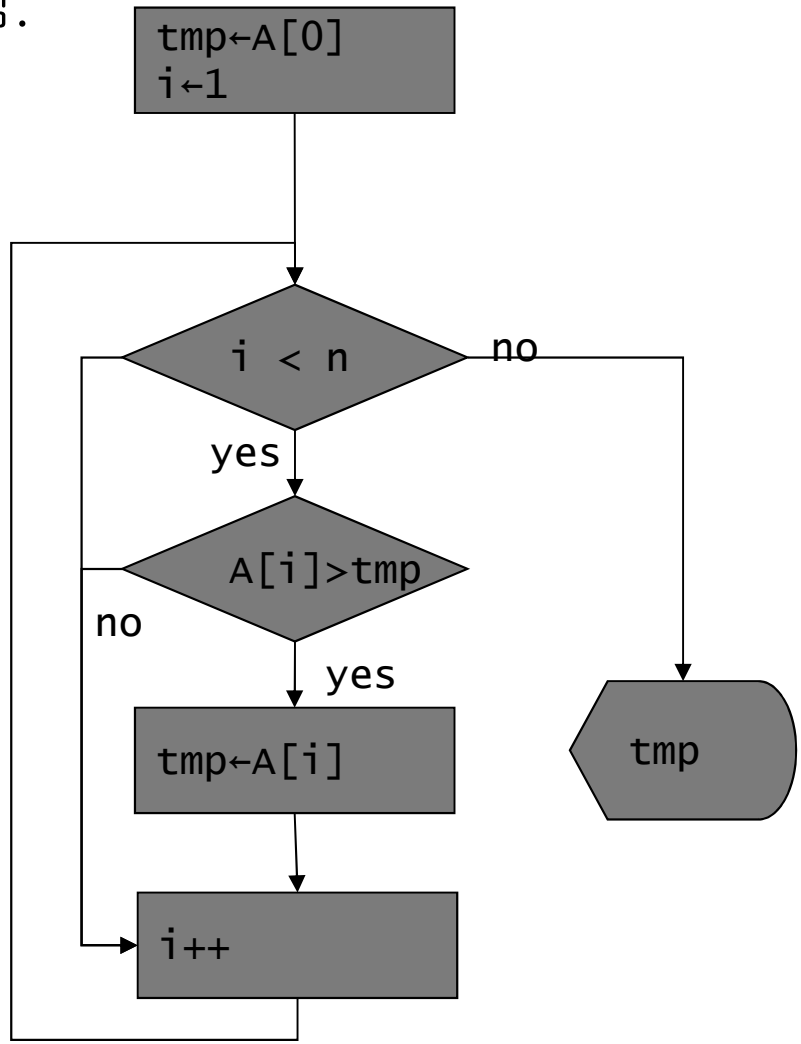
- (예) 배열에서 최대값 찾기 알고리즘

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

흐름도로 표기된 알고리즘

- 특징
  - 직관적이고 이해하기 쉬운 알고리즘 기술 방법
  - 복잡한 알고리즘의 경우, 상당히 복잡해짐.



유사코드로 표현된 알고리즘

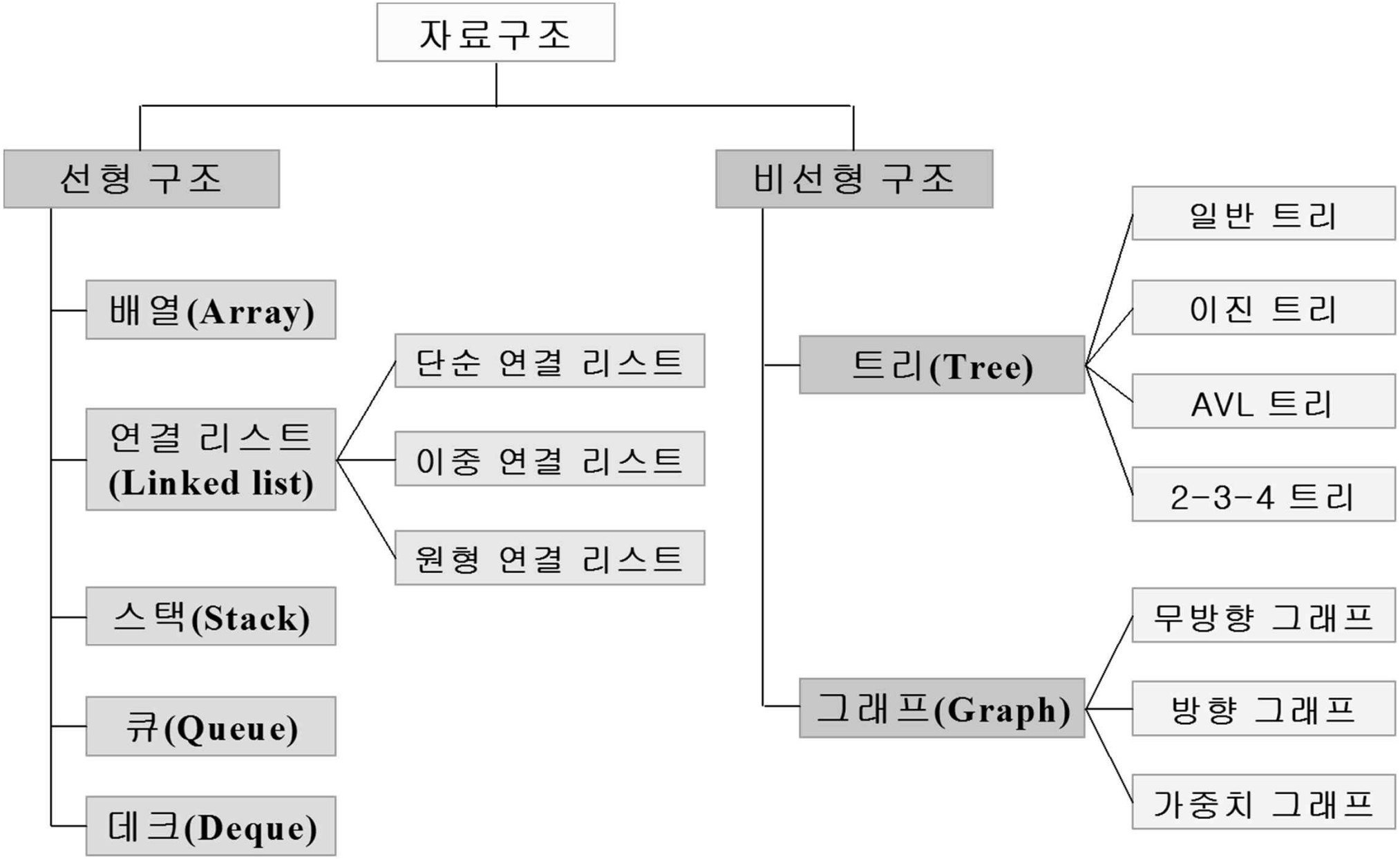
- 특징
  - 알고리즘의 고수준 기술 방법
  - 자연어보다는 더 구조적인 표현 방법
  - 프로그래밍 언어보다는 덜 구체적인 표현방법
  - 알고리즘 기술에 가장 많이 사용
  - 프로그램을 구현할 때의 여러가지 문제들을 감출 수 있다. 즉 알고리즘의 핵심적인 내용에만 집중할 수 있다.

C로 표현된 알고리즘

- 특징
  - 알고리즘의 가장 정확한 기술이 가능
  - 반면 실제 구현시의 많은 구체적인 사항들이 알고리즘의 핵심적인 내용들의 이해를 방해할 수 있다.

```
#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;
}
```

자료구조의 분류


자료구조
선형 구조
비선형 구조
배열(Array)
연결 리스트 (Linked list)
단순 연결 리스트
이중 연결 리스트
원형 연결 리스트
스택(Stack)
큐(Queue)
데크(Deque)
트리(Tree)
일반 트리
이진 트리
AVL 트리
2-3-4 트리
그래프(Graph)
무방향 그래프
방향 그래프
가중치 그래프

알고리즘의 성능분석

- 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 n의 함수
    - **시간 복잡도 분석**: 수행 시간 분석
    - **공간 복잡도 분석:** 수행시 필요로 하는 메모리 공간 분석

수행시간측정

- 컴퓨터에서 수행시간을 측정하는 방법에는 주로 clock 함수 사용.

- clock_t clock(void);
  - clock 함수는 호출되었을 때의 시스템 시각을 CLOCKS_PER_SEC 단위로 반환

```
#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);
}
```

복잡도 분석

- 시간 복잡도는 알고리즘을 이루고 있는 연산들이 몇 번이나 수행되는지를 숫자로 표시
- 산술, 대입, 비교, 이동 연산의 기본적인 연산:
  - 수행시간이 입력의 크기에 따라 변하면 안됨
- 알고리즘이 수행하는 연산의 개수를 계산하여 두개의 알고리즘을 비교할 수 있다.
- 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n에 대한 함수->**시간복잡도 함수**라고 하고 T(n) 이라고 표기한다.

복잡도 분석의 예

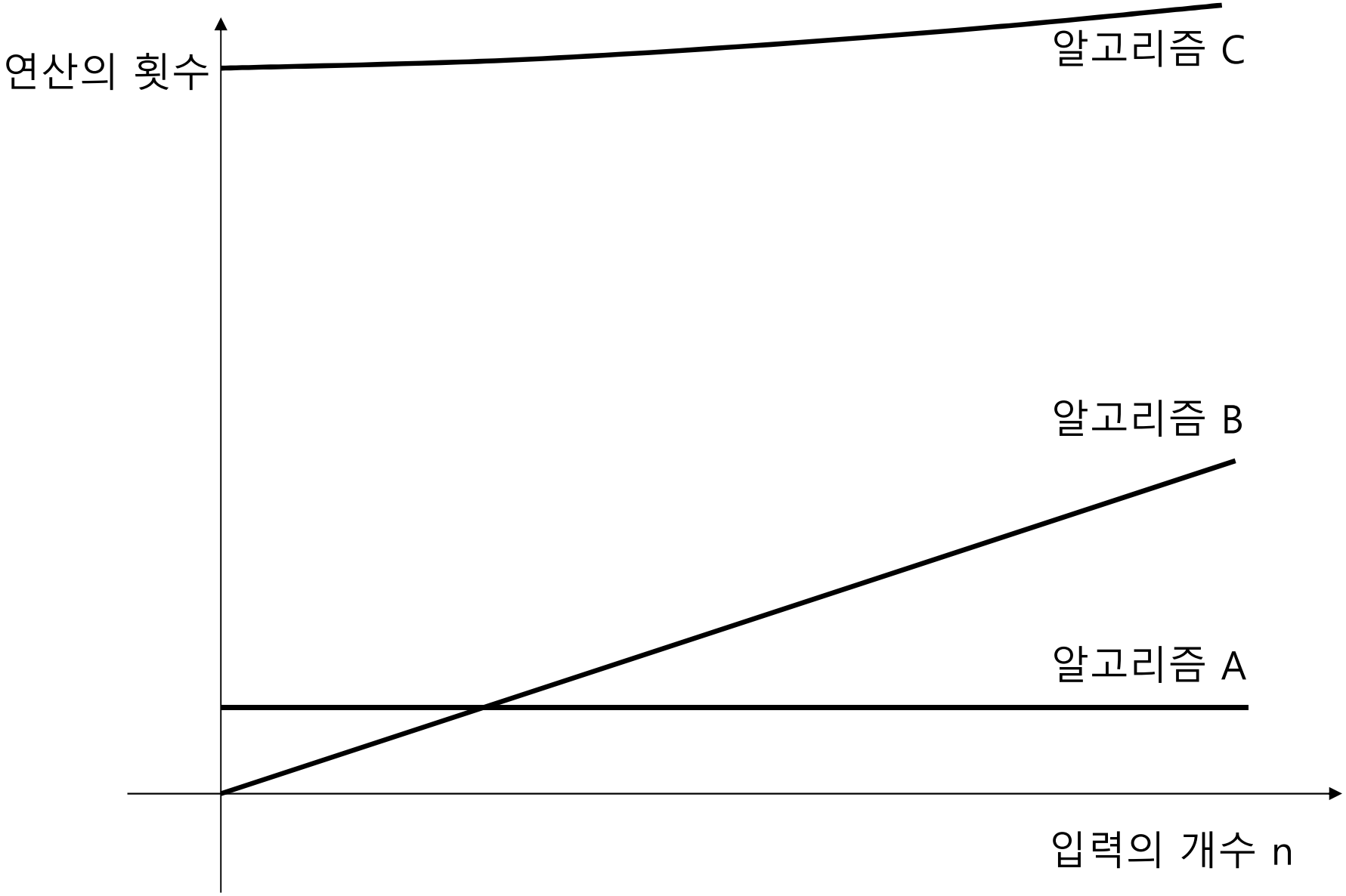
- n을 n번 더하는 문제:
  - 각 알고리즘이 수행하는 연산의 개수를 세어 본다.
  - 단 for 루프 제어 연산은 고려하지 않음.

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	$2n^2 + 1$

연산의 횟수를 그래프로 표현



시간복잡도 함수 계산 예

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

```
ArrayMax(A,n)

  tmp ← A[0];                          1번의 대입 연산
  for i←1 to n-1 do                    루프 제어 연산은 제외
          if tmp < A[i] then           n-1번의 비교 연산
                  tmp ← A[i];          n-1번의 대입 연산(최대)
  return tmp;                          1번의 반환 연산

                                       총 연산수= 2n(최대)
```

빅오 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 의 값이 전체의 약 99%인 1,000,000이고 두 번째 항의 값이 1000으로 전체의 약 1%를 차지한다.
- 보통 시간복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분하다.

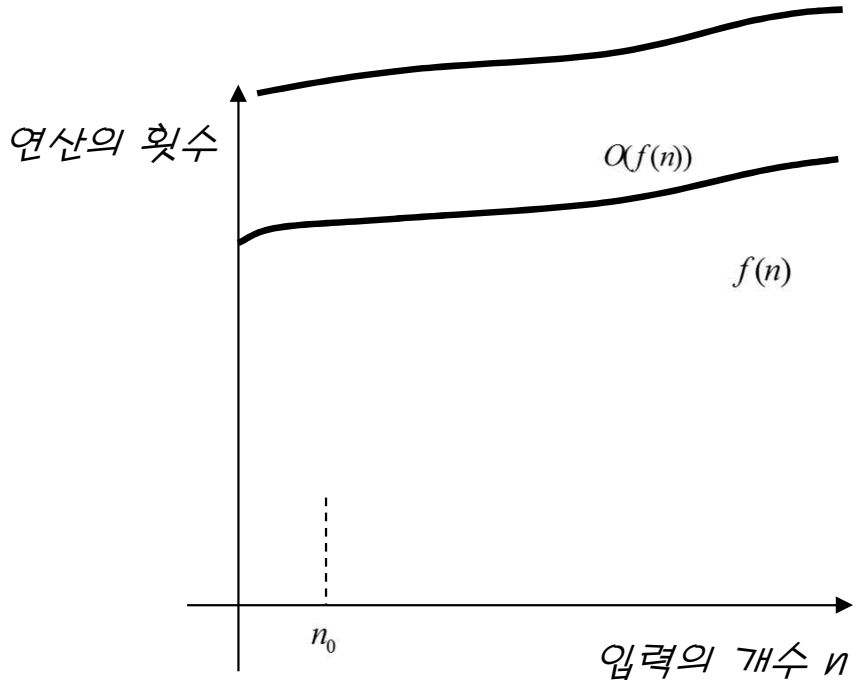
- 빅오표기법: 연산의 횟수를 대략적(점근적)으로 표기한 것
  - 두개의 함수 f(n)과 g(n)이 주어졌을 때,
  - 모든 $n \geq n0$에 대하여 $|f(n)| \leq c|g(n)|$을 만족하는 2개의 상수 c와 n0가 존재하면 f(n)=O(g(n))이다.
- 빅오는 함수의 상한을 표시한다.
  - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $2n+1 = O(n)$

n=1000인 경우

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

99%

1%



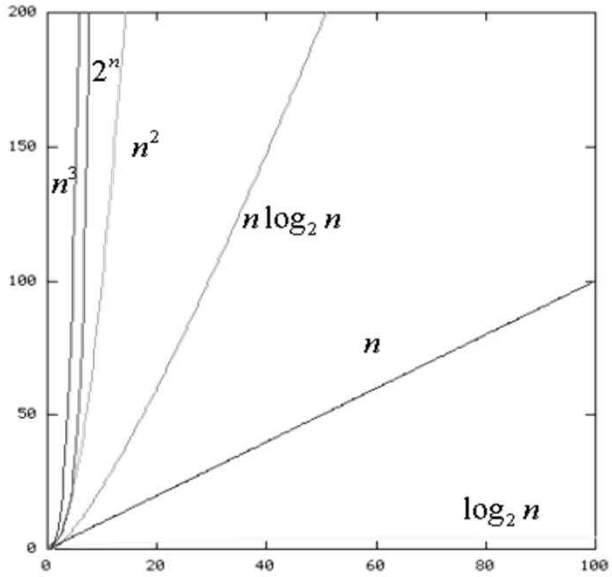
빅오 표기법의 예

예제 1.1 빅오 표기법

- $f(n)=5$이면 $O(1)$이다. 왜냐하면 $n_0=1$, $c=10$일 때, $n \geqq 1$에 대하여 $5 \leqq 10 \cdot 1$이 되기 때문이다.
- $f(n)=2n+1$이면 $O(n)$이다. 왜냐하면 $n_0=2$, $c=3$일 때, $n \geqq 2$에 대하여 $2n+1 \leqq 3n$이 되기 때문이다.
- $f(n)=3n^2+100$이면 $O(n^2)$이다. 왜냐하면 $n_0=100$, $c=5$일 때, $n \geqq 100$에 대하여 $3n^2+100 \leqq 5n^2$이 되기 때문이다.
- $f(n)=5 \cdot 2^n+10n^2+100$이면 $O(2^n)$이다. 왜냐하면 $n_0=1000$, $c=10$일 때, $n \geqq 1000$에 대하여 $5 \cdot 2^n+10n^2+100 \leqq 10 \cdot 2^n$이 되기 때문이다.

빅오 표기법의 종류

- O(1) : 상수형
- O(logn) : 로그형
- O(n) : 선형
- O(nlogn) : 로그선형
- O(n2) : 2차형
- O(n3) : 3차형
- O(nk) : k차형
- O(2n) : 지수형
- O(n!) : 팩토리얼형



시간복잡도	*N*					
	1	2	4	8	16	32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	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}

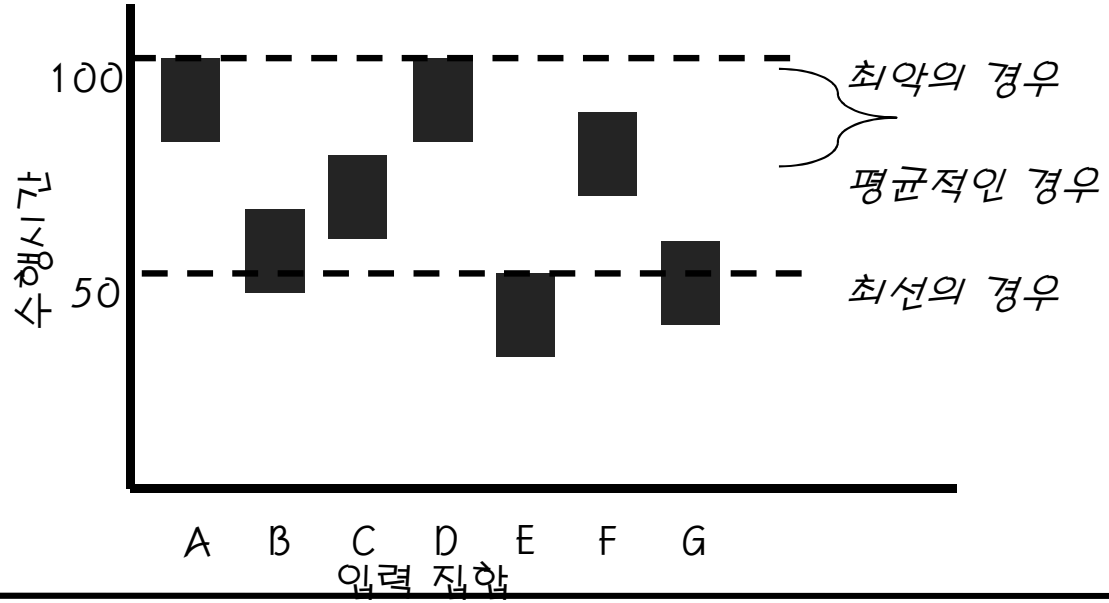
빅오 표기법이외의 표기법

- 빅오메가 표기법
  - 모든 n≥n0에 대하여 |f(n)| ≥ c|g(n)|을 만족하는 2개의 상수 c와 n0가 존재하면 f(n)=Ω(g(n))이다.
  - 빅오메가는 함수의 하한을 표시한다.
  - (예) n ≥ 5 이면 2n+1 <10n 이므로 n = Ω(n)

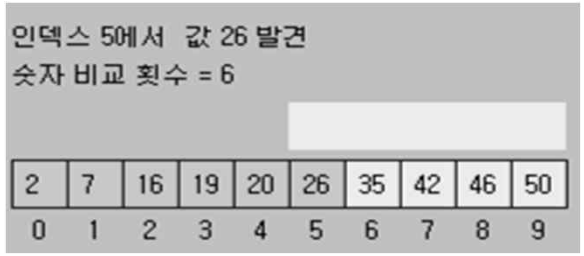
- 빅세타 표기법
  - 모든 n≥n0에 대하여 c1|g(n)| ≤ |f(n)| ≤ c2|g(n)|을 만족하는 3개의 상수 c1, c2와 n0가 존재하면 f(n)=θ(g(n))이다.
  - 빅세타는 함수의 하한인 동시에 상한을 표시한다.
  - f(n)=O(g(n))이면서 f(n)= Ω(g(n))이면 f(n)= θ(n)이다.
  - (예) n ≥ 1이면 n ≤ 2n+1 ≤ 3n이므로 2n+1 = θ(n)

최선, 평균, 최악의 경우

- 알고리즘의 수행시간은 입력 자료 집합에 따라 다를 수 있다.
  (예) 정렬 알고리즘의 수행 시간은 입력 집합에 따라 다를 수 있다
  - 최선의 경우(best case)
    - 수행 시간이 가장 빠른 경우, 의미가 없는 경우가 많다
  - 평균의 경우(average case)
    - 수행시간이 평균적인 경우, 계산하기가 상당히 어려움
  - 최악의 경우(worst case)
    - 수행 시간이 가장 늦은 경우, 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.



- 순차탐색
  - **최선의 경우**: 찾고자 하는 숫자가 맨앞에 있는 경우
    - ∴ O(1)
  - **최악의 경우**: 찾고자 하는 숫자가 맨뒤에 있는 경우
    - ∴ O(n)
  - **평균적인 경우**: 각 요소들이 균일하게 탐색된다고 가정하면
    - (1+2+…+n)/n=(n+1)/2, ∴ O(n)



자료구조 기술규칙

- 상수
  - 대문자로 표기, (예) #define MAX_ELEMENT 100

- 변수 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

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

(예) typedef int element;

```
typedef struct ListNode {
    element data;
    struct ListNode *link;
} ListNode;
```