DATA
STRUCTURES
USING C

학습목표

- 자료구조와 알고리즘의 개념을 이해한다
- 알고리즘의 다양한 표기 방법을 이해한다
- 추상화의 필요성과 추상 자료형의 개념을 이해한다.
- 알고리즘의 실행 시간 측정 방법을 이해한다.

01 CHAPTER 자료구조와 알고리즘

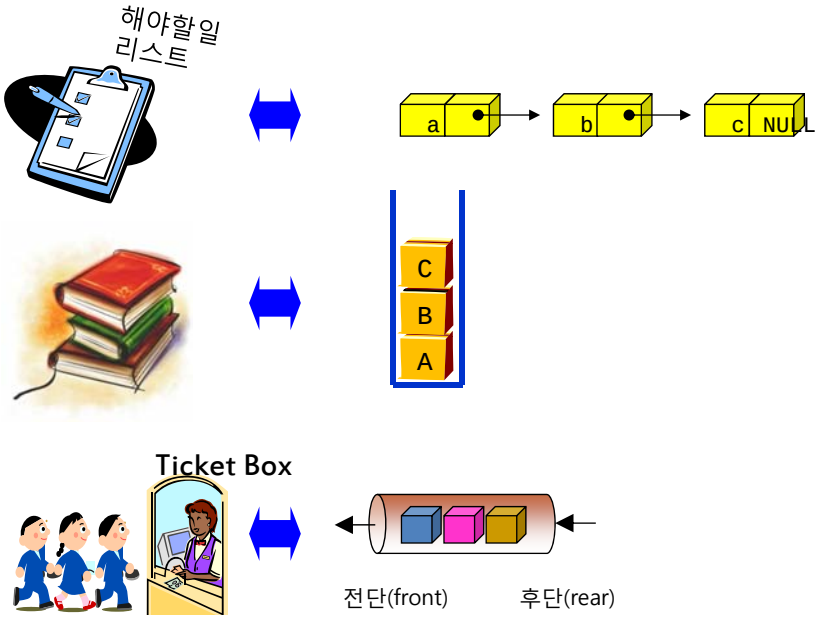
1.1 자료구조

- 일상 생활에서 자료를 정리하고 조직화하는 이유는?
  - 사물을 편리하고 효율적으로 사용하기 위함
  - 다양한 자료를 효율적인 규칙에 따라 정리한 예

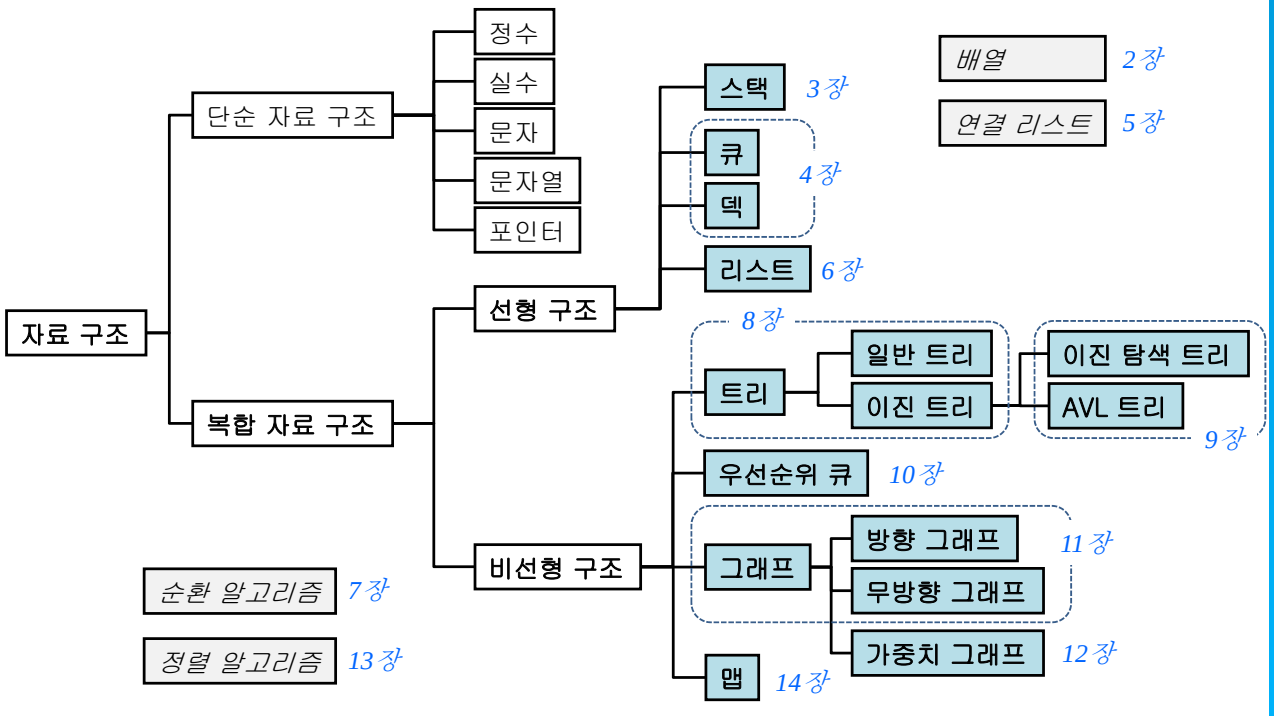
컴퓨터에서의 자료구조

- 자료구조(Data Structure)
  - 컴퓨터에서 자료를 정리하고 조직화하는 다양한 구조

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



자료구조의 분류

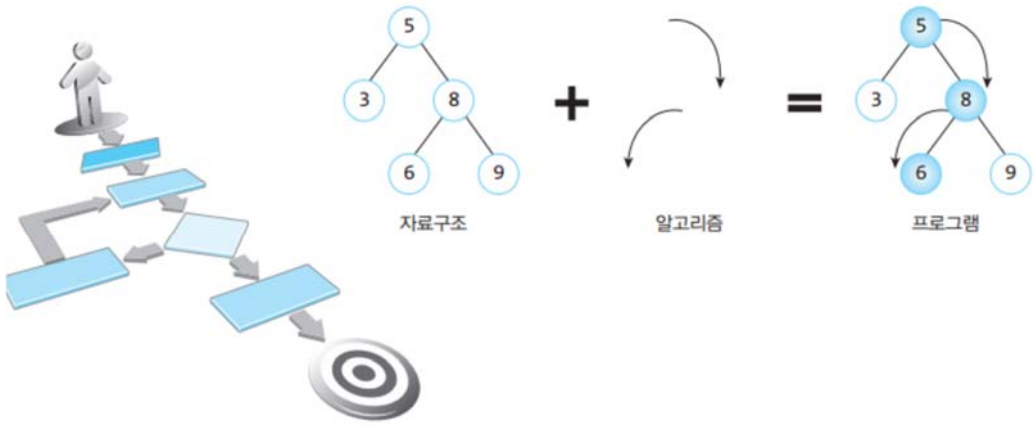


- 다음 중 선형 자료구조로 볼 수 없는 것은?

1. 스택
2. 큐
3. 트리
4. 리스트

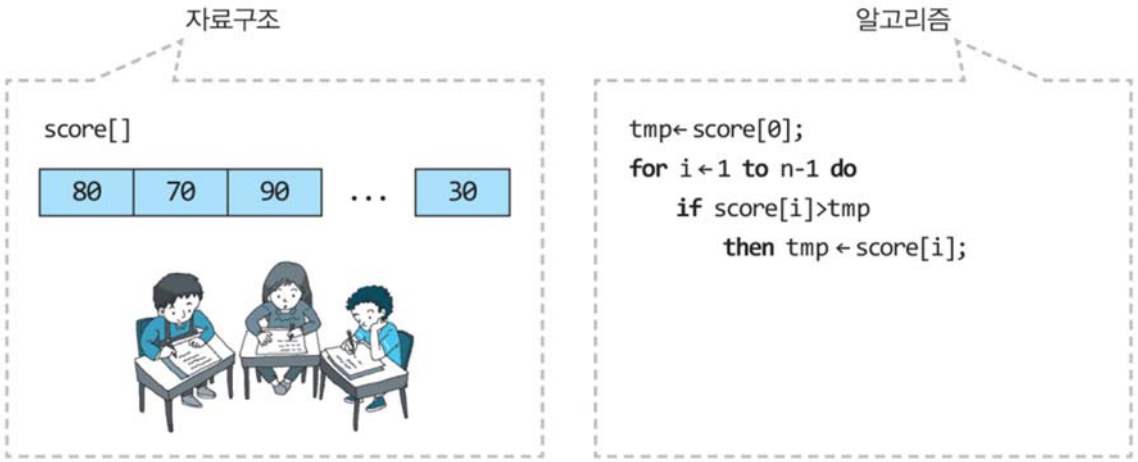
컴퓨터 프로그램의 구성

- 컴퓨터 프로그램은 무엇으로 이루어져 있나?
  - **프로그램** = **자료구조** + **알고리즘**



| 그림 1.3 알고리즘은 문제를 해결하는 절차

- (예) 최대값 탐색 프로그램 = 배열+ 순차탐색



알고리즘

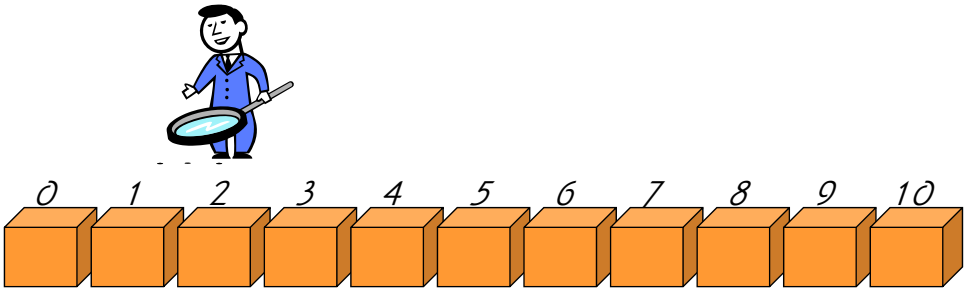
- *컴퓨터로 문제를 풀기 위한 단계적인 절차*
  - 예) 전화번호부에서 특정 사람 이름 찾기

- 알고리즘의 조건
  - 입 력 : 0개 이상의 입력이 존재하여야 한다.
  - 출 력 : 1개 이상의 출력이 존재하여야 한다.
  - 명백성 : 각 명령어의 의미는 모호하지 않고 명확해야 함.
  - 유한성 : 한정된 수의 단계 후에는 반드시 종료되어야 함.
  - 유효성 : 각 명령어들은 실행 가능한 연산이어야 한다.

알고리즘의 기술 방법

- 방법들
  1. 영어나 한국어와 같은 자연어
  2. 흐름도(flow chart)
  3. 유사 코드(pseudo-code)
  4. 특정한 프로그래밍 언어 (C언어, C++, java 등)

- (예) n개 정수배열에서 최대값 찾기 알고리즘



알고리즘의 기술 방법(1)

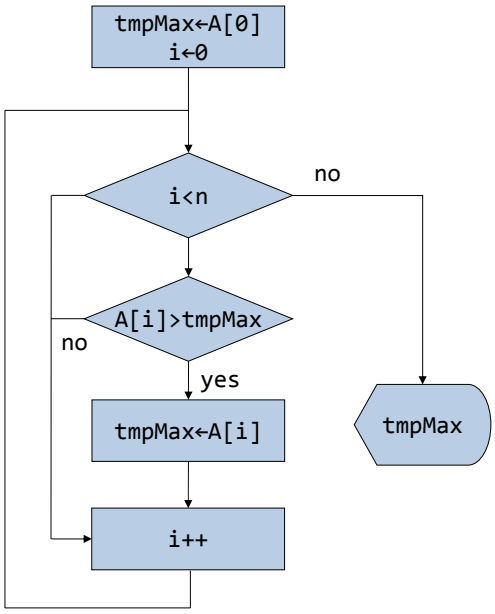
(1) 자연어로 표기된 알고리즘
- 인간이 읽기가 쉽다.
- 단어들을 정확하게 정의하지 않으면 의미 전달이 모호해질 우려가 있다.

```
ArrayMax(A,n)

1. 배열 A의 첫 번째 요소를 변수 tmp에 복사
2. 배열 A의 다음 요소들을 차례대로 tmp와 비교하여 더 크면 tmp로
   복사
3. 배열 A의 모든 요소를 비교했으면 tmp를 반환
```

알고리즘의 기술 방법(2)

(2) 흐름도(flowchart)로 표기된 알고리즘

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



알고리즘의 기술 방법(3)

(3) **유사코드**로 표현된 알고리즘

- 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 있음
- 알고리즘의 핵심적인 내용에만 집중할 수 있음

```
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 =는 비교연산

알고리즘의 기술 방법(4)

(4) **특정 프로그래밍 언어**로 표현된 알고리즘

- 알고리즘의 가장 정확한 기술 가능
- 실제 구현시의 많은 구체적인 사항들이 알고리즘의 핵심적인 내용들의 이해를 방해할 수 있음
- 예) C언어로 표기된 알고리즘

```
#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;
}
```


1.3 자료형과 추상 자료형

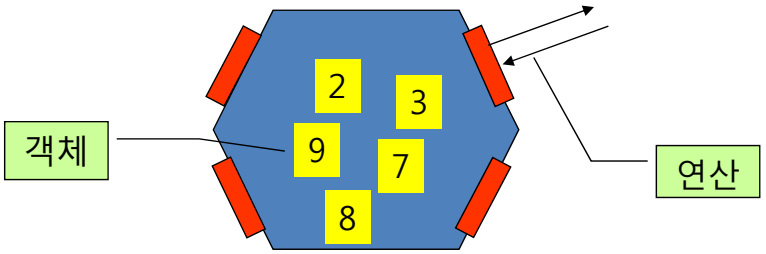
- 추상화란?
  - 어떤 시스템의 간략화 된 기술 또는 명세
  - 시스템의 정말 핵심적인 구조나 동작에만 집중
- 자료형(data type)
  - 데이터의 집합과 연산의 집합

예) int 데이터 타입 { 데이터: { ...,-2,-1,0,1,2,...}
연산: +, -, /, *, %

- 추상 자료형(ADT: Abstract Data Type)
  - 데이터 타입을 추상적(수학적)으로 정의한 것
  - 데이터나 연산이 **무엇(what)**인가를 정의함
  - 데이터나 연산을 **어떻게(how)** 구현할 것인지는 정의하지 않음

추상 자료형의 정의

- 데이터+연산



- **자연수 ADT**

데이터: 1에서 시작하여 INT_MAX까지의 순서화된 정수의 부분 범위

연산:

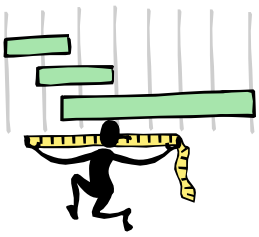
- add(x,y): x+y가 INT_MAX보다 작으면 x+y를 반환
- distance(x,y): x가 y보다 크면 x-y를 반환하고 작으면 y-x를 반환
- equal(x,y): x와 y가 같은 값이면 TRUE, 아니면 FALSE를 반환
- successor(x): x가 INT_MAX보다 작으면 x+1을 반환한다.

추상 자료형과 자료구조

- 추상 자료형과 자료구조의 관계
  - 자료 구조는 추상 자료형을 프로그래밍 언어로 구현한 것
  - 먼저 각 자료 구조의 추상 자료형을 소개
  - 추상 자료형의 구현 ➔ 이 책에서는 C언어를 사용
    - 추상 자료형의 데이터: 구조체를 사용
    - 추상 자료형의 연산: 함수를 사용
  - 다른 방법: 객체지향언어(C++, Java 등)

1.4 알고리즘의 성능분석

- 알고리즘의 성능 분석 기법
  - **실행 시간을 측정하는 방법**
    - 두 개의 알고리즘의 실제 실행 시간을 측정하는 것
    - 실제로 구현하는 것이 필요
    - 동일한 하드웨어, 소프트웨어를 사용하여야 함
    - 다른 데이터로는 다른 결과가 나올 수 있음
  - **알고리즘의 복잡도를 분석하는 방법**
    - 직접 구현하지 않고서도 효율성을 분석하는 것
      - **시간 복잡도 분석** : 실행 시간 분석
        - » 알고리즘이 실행하는 연산의 횟수를 측정하여 비교
        - » 일반적으로 연산의 횟수는 n의 함수로 나타냄
      - **공간 복잡도 분석** : 실행시 필요로 하는 메모리 공간 분석



(1) 실행시간 측정

- clock 함수 사용: clock_t clock(void);
  - 호출되었을 때의 시스템 시각 반환
  - CLOCKS_PER_SEC 단위
  - 실행 시간 측정 프로그램 예

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

(2) 복잡도 분석

- 시간 복잡도
  - 산술, 대입, 비교, 이동의 기본적인 연산 고려
  - 알고리즘 실행에 필요한 연산의 개수를 계산
  - 입력의 개수 n에 대한 함수->**시간복잡도 함수**, T(n)

복잡도 분석의 예

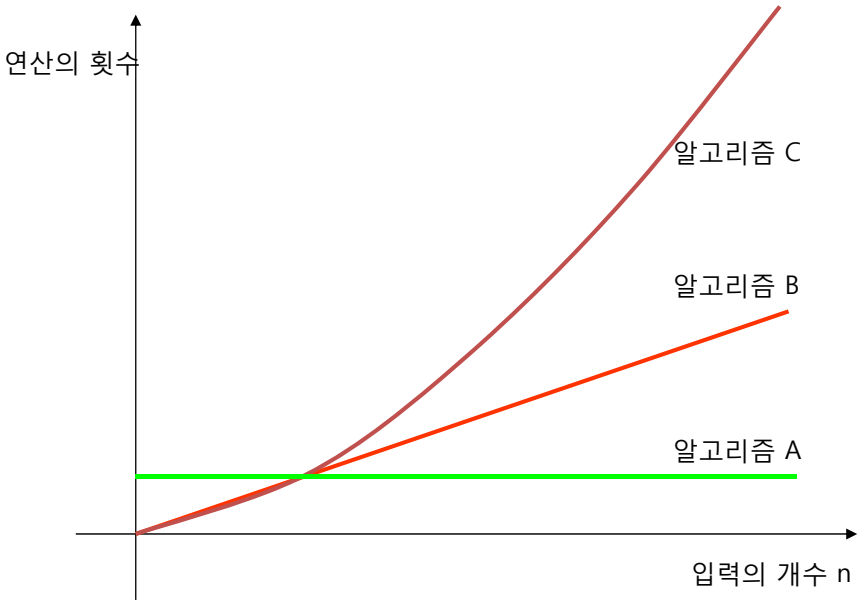
- n을 n번 더하는 문제
  - 각 알고리즘이 수행하는 연산의 개수 계산
  - 단 for 루프 제어 연산은 고려하지 않음

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 j←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



시간 복잡도 함수 계산 예

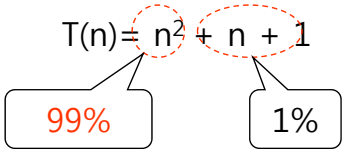
- 코드를 분석해보면 수행되는 연산의 횟수를 입력 크기의 함수로 만들 수 있다.

```
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

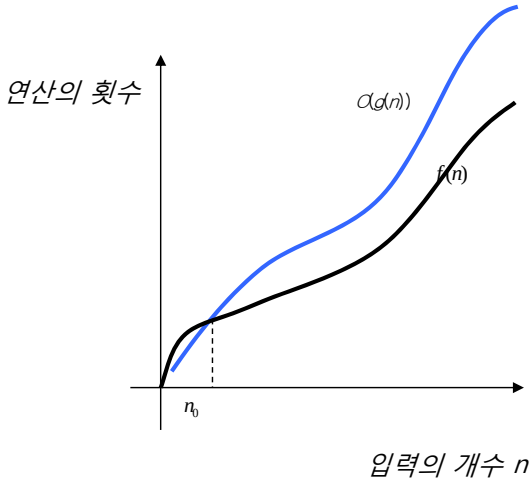
- 차수가 가장 큰 항이 가장 영향을 크게 미치고 다른 항들은 상대적으로 무시될 수 있음
  - 예: T(n) = $n^2 + n + 1$
    - n=1일때 : T(n) = 1 + 1 + 1 = 3 (33.3%)
    - n=10일때 : T(n) = 100 + 10 + 1 = 111 (90%)
    - n=100일때 : T(n) = 10000 + 100 + 1 = 10101 (99%)
    - n=1,000일때 : T(n) = 1000000 + 1000 + 1 = 1001001 (99.9%)

n=100인 경우



빅오 표기법의 정의

- 두 개의 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 f(n)=O(g(n))이다.
  - 연산의 횟수를 대략적(점근적)으로 표기한 것
  - 빅오는 함수의 상한을 표시한다.
  - (예) n≥5 이면 2n+1 <10n 이므로 2n+1 = O(n)



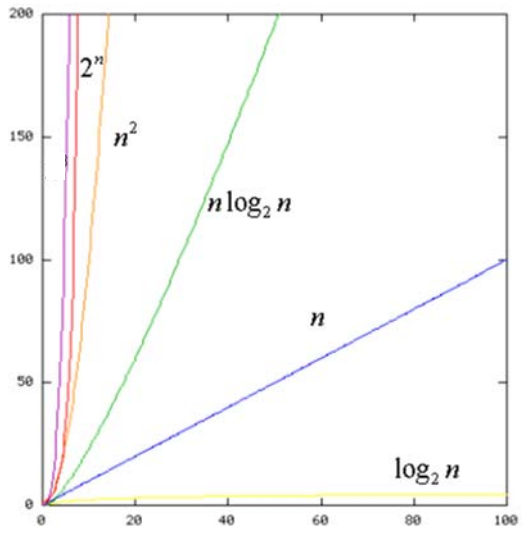
빅오 표기법의 예

- 예제 1.1 빅오 표기법
  - $f(n)$=5이면 $O(1)$이다. 왜냐하면 n_0=1, c=10일 때, $n \geq 1$에 대하여 $5 \leq 10 \cdot 1$이 되기 때문이다.
  - $f(n)$=2n+1이면 $O(n)$이다. 왜냐하면 n_0=2, c=3일 때, $n \geq 2$에 대하여 $2n+1 \leq 3n$이 되기 때문이다.
  - $f(n)$=$3n^2+100$ 이면 $O(n^2)$이다. 왜냐하면 n_0=100, c=5일 때, $n \geq 100$에 대하여 $3n^2+100 \leq 5n^2$이 되기 때문이다.
  - $f(n)$=$5 \cdot 2^n + 10n^2+100$ 이면 $O(2^n)$이다. 왜냐하면 n_0=1000, c=10일 때, $n \geq 1000$에 대하여 $5 \cdot 2^n + 10n^2+100 < 10 \cdot 2^n$이 되기 때문이다.

- 30n+4 가 O(n)임을 증명하라. O(n)의 정의를 이용한다.

빅오 표기법의 종류

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



빅오 표기법의 종류

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}

- 시간 복잡도 함수가 7n+10 이라면 이것이 나타내는 것은 무엇인가?
  ① 연산의 횟수
  ② 프로그램의 컴파일 시간
  ③ 프로그램이 차지하는 메모리의 양
  ④ 입력 데이터의 총 개수

- $O(n^2)$의 시간 복잡도를 가지는 알고리즘에서 입력의 개수가 2배로 되었다면 실행 시간은 어떻게 되는가?
  ① 변함없다.
  ② 2배
  ③ 4배
  ④ 8배

- 동일한 문제를 해결하는 알고리즘 A, B, C, D의 시간 복잡도가 다음과 같이 계산되었다고 하자. n이 충분히 크다고 할 때, 실행 시간이 적은 것부터 순서대로 나열하라.

  A: $O(n)$ B: $O(n^2)$ C: $O(n\log n)$ D: $O(2^n)$

- 다음의 시간 복잡도 함수를 빅오 표기법으로 표시하라.

  ① $9n^2 + 8n + 1$

  ② $n! + 2^n$

  ③ $n^2 + n\log_2 n + 1$

  ④ $\sum_{i=1}^{n} i^2$

- 다음 코드의 시간 복잡도를 구하라.

```
int i, j;
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        printf("%d %d ₩n", i, j);
    }
}
```


- 다음 코드에서 대입 연산, 덧셈 연산, 비교 연산의 개수를 계산하여 시간 복잡도 함수를 계산하라.

```
float sum(float list[], int n)
{
        float tempsum;
        int i;
        tempsum = 0;
        for(i=0;i<n;i++) {
                tempsum += list[i];
        }
        tempsum += 100;
        tempsum += 200;
        return tempsum;
}
```


34

- 다음 프로그램의 시간 복잡도를 빅오 표기법으로 나타내라.

```
for (i=1; i<n; i *= 2)
    ++k;
```

- sub 함수의 시간 복잡도가 O(n)일 때 다음 문장의 시간 복잡도는?

```
for (i=1; i<n; i *= 2)
    sub();
```


- 다음 알고리즘의 시간 복잡도를 n에 대한 함수로 나타내고, 빅오 표기법으로도 나타내어라.

```
int algorithm(int n) {
    int k = 0;
    while (n > 1) {
        n = n/2;
        k++;
    }
    return k;
}
```

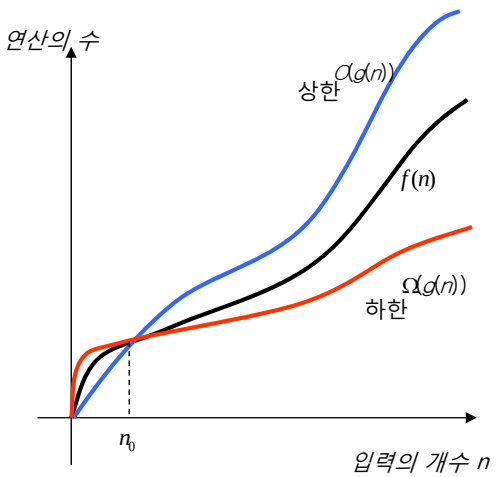
- 다음 프로그램의 코드에 대하여 답하라.

```
int i, k;
for (i=0; i<(n-2); i++) {
    for (k=0; k<30; k++) {
        buffer[i][k] = 0;
    }
}
```

1. 시간 복잡도를 n에 대한 함수로 나타내고, 빅오 표기법으로도 나타내어라. 여기서 입력의 개수는 양의 정수 n이다.
2. 위의 프로그램에서 입력의 개수도 100배 증가하였고 CPU의 속도도 100배 증가하였을 경우, 위의 프로그램의 실행 시간이 늘어나는가? 아니면 줄어드는가? 그 이유는?

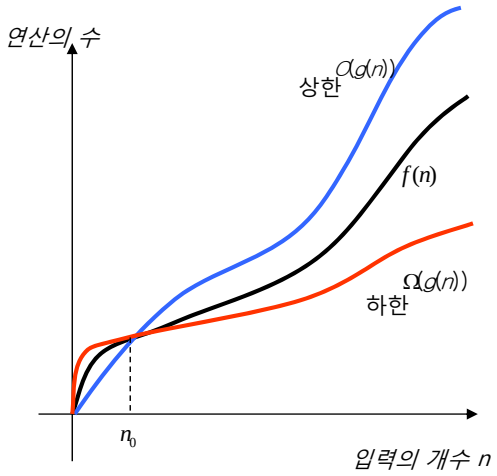
빅오메가 표기법

- 정의
  - 모든 $n \geq n_0$에 대하여 $|f(n)| \geq c|g(n)|$을 만족하는 2개의 상수 c와 n_0가 존재하면 $f(n)=\Omega(g(n))$이다.
  - 빅오메가는 함수의 하한을 표시
  - (예) $n \geq 1$ 이면 $2n+1 > n$ 이므로 $n = \Omega(n)$



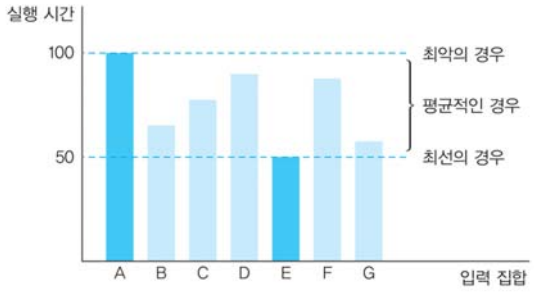
빅세타 표기법

- 정의
  - 모든 $n \geq n_0$에 대하여 $c1|g(n)| \leq |f(n)| \leq c2|g(n)|$을 만족하는 3개의 상수 c1, c2와 n_0가 존재하면 $f(n)=\theta(g(n))$이다.
  - 빅세타는 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 $2n+1 = \theta(n)$



최선, 평균, 최악의 경우

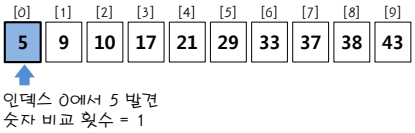
- 실행시간은 입력 집합에 따라 다를 수 있음
  - 최선의 경우(best case): 수행 시간이 가장 빠른 경우
    - 의미가 없는 경우가 많다.
  - 평균의 경우(average case): 수행시간이 평균적인 경우
    - 계산하기가 상당히 어려움.
  - **최악의 경우(worst case)**: 수행 시간이 가장 늦은 경우
    - 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.
      - (예) 비행기 관제업무, 게임, 로보틱스



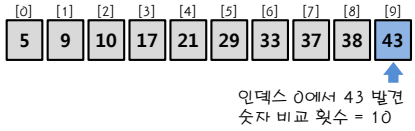
예 : 순차탐색의 최선, 평균, 최악

```
int sequentialSearch(int list[], int n, int key) {
        for(int i=0 ; i<n ; i++)
                // 탐색에 성공하면 키 값의 인덱스 반환
                if(list[i]==key) return i;
        return -1;      // 탐색에 실패하면 -1 반환
}
```

- 최선의 경우: O(1)



- 최악의 경우: O(n)



- 평균적인 경우: (1+2+…+n)/n=(n+1)/2 ∴ O(n)

실습

- 1부터 n까지의 합을 구하는 방법은 다음과 같이 3가지가 있다.

  알고리즘 A: sum = n(n+1)/2 공식 사용

  알고리즘 B: sum = 1+2+…+n

  알고리즘 C: sum = 0+(1)+(1+1)+(1+1+1)+…+(1+1+…+1)

  1. 각 알고리즘을 함수로 구현하라. n을 매개 변수로 전달받고 결과를 반환한다.
  2. 비교적 작은 n에 대해 각 함수를 호출하여 세 알고리즘의 계산 결과가 동일함을 확인하라.
  3. 세 알고리즘의 시간 복잡도를 이론적으로 분석해 보고 빅오 표기법으로 나타내라.
  4. 각 알고리즘의 실제 실행시간을 측정하여 이론적인 분석과 같게 나오는지를 조사하라. 실행 시간 측정을 위해 한 번의 루프에서 n은 2000 정도씩 증가시키고, 알고리즘 C의 실행시간이 1초 이하일 때까지 실행시킨 결과를 표로 만들고 선 그래프로 그려라 (엑셀을 사용)
  5. C알고리즘을 제외하고 한 번의 루프에서 n을 더 크게 증가시켜 알고리즘 B가 1초 이하일 때까지 실행시킨 결과를 표로 만들고, 그래프로 그려라. 결과가 이론적인 분석과 같은지 조사하라.

과제

– 풀이를 .docx 문서로 eclass에 제출