SCH 순천향대학교

자료구조 1

2018학년도 1학기

담당교수: 홍 민

수업 진행

- 목적: 프로그램에 꼭 필요한 자료구조의 개념과 기본적인 알고리즘들을 학습하며 프로그램의 효율성을 분석하고 향상시킬 수 있는 방법을 배움
- 교재: C 언어로 쉽게 풀어쓴 자료 구조, 천인국, 공용해, 하상호 저, 생능출판사, 2014 개정판
- 성적: 중간고사 30%, 기말고사 30%, 과제물 30%, 퀴즈 10%
- 강의노트: http://cyber.sch.ac.kr
- 강의 방법 및 report: 강의 노트를 따라 진행. 과제는 매주에 한번 정도 프로그래밍과 문제 형식
- 절대 Cheating은 불가!!!

수업 진행

- 전체 수업의 1/3 이상 결석 시 무조건 F임
- 2번 지각하면 1번의 결석으로 처리
- 과제는 항상 자기 혼자의 힘으로 할 것
- 예습 및 복습을 철저히 할 것
- 항상 수업에 최선을 다하는 자세가 중요
- 모든 과제는 Visual Studio 6.0을 이용할 것!!
- **절대 Cheating은 불가!!!**

파일 입출력

- FILE 구조체를 사용, 선언

  FILE *fp;

- 파일 열기(읽기모드)

  fp = fopen("파일 이름", "모드");

- 파일 닫기

  fclose(fp);

FILE *fopen(const char *filename, const char *mode);

- 리턴값
  - 지정한 파일을 지정한 모드로 열고 입출력에 필요한 FILE구조체를 생성한 후 그 포인터를 리턴하지만 에러가 발생하면 NULL을 리턴

- 파일 이름(filename)
  - 액세스할 대상 파일의 이름을 넣는 부분
  - 파일의 경로를 지정할 때
    - C:₩Data₩File.txt – X
    - C:₩₩Data₩₩File.txt – O
  - 경로나 파일명은 “ ” 로 묶어줘야 한다.

FILE *fopen(const char *filename, const char *mode);

- 모드
  - r : 읽기 전용, 파일이 없으면 에러를 리턴
  - w : 쓰기 전용, 파일이 없으면 새로 만들고 있으면 기존의 파일을 지우고 빈 파일을 생성
  - a : 추가 모드, 파일의 끝부터 데이터를 추가, 파일이 없으면 파일 생성
  - r+ : 읽고 쓰기 가능, 파일이 없으면 에러를 리턴
  - w+ : 읽고 쓰기 가능, 파일이 없으면 새로 생성
  - a+ : 읽고 추가 가능, 파일이 없으면 새로 생성

  - 모드는 “ ” 로 묶어줘야 한다.

int fclose(FILE *stream);

- FILE 구조체를 해제
- 열린 파일은 이 함수로 닫아야 한다.

```
fclose(fp);
```

int feof(FILE *stream);

- 파일의 끝까지 계속 반복하기 위해 사용
- 파일의 끝이 아니라면 0을, 파일의 끝이라면 0이 아닌 수를 리턴

```
while(!feof(fp))
{
    fscanf(fp, "%d", &nTemp);
}
```

  - 파일의 정수형을 하나씩 읽어 nTemp에 저장. 위 경우 파일의 마지막 숫자가 nTemp에 저장되면서 while문이 종료.

입출력 함수들

- int fscanf(FILE *stream, const char *format [, argument]...);
  - scanf함수와 동일한 방법으로 사용된다.

fscanf(fp, “%d%d”, &nNum1, &nNum2);

-> 파일에서 정수2개를 읽어 nNum1과 nNum2변수에 저장

- int fprintf(FILE *stream, const char *format [, argument]...);
  - printf함수와 동일한 방법으로 사용된다.

fprintf(fp, “%d%d”, nNum1, nNum2);

-> 파일에 정수nNum1과 nNum2를 쓴다.

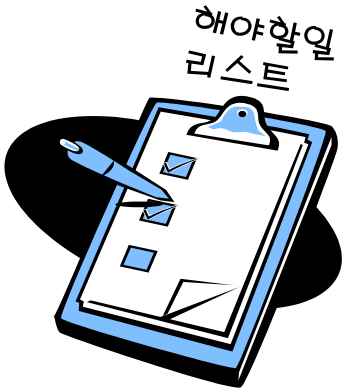
```c
#include <stdio.h>

double show(double, char, double);

int main()
{
    FILE *fp;   // 파일포인터 변수선언
    char oper;
    double a, b, res;

    fp = fopen("data.txt", "r");    // data.txt파일을 읽기모드로 열어 생성된 FILE구조체를 fp에 할당
    if(fp==NULL)    // 만약 fopen함수에서 에러가 발생하여 fp에 NULL값이 저장되었다면 파일열기실패
    {
        printf("파일에 열리지 않았습니다.₩n");
        return 0;   // 메세지 출력하고 프로그램 종료
    }

    while(!feof(fp))    // 파일포인터가 파일의 끝을 가르킬때까지 반복한다.
    {
        fscanf(fp, "%lf%c%lf", &a, &oper, &b);  // 파일에서 실수형 문자형 실수형으로 값일 읽어
                                                // 실수형을 a, b에 문자형은 oper에 저장

        res = show(a, oper, b); // 입력받은 숫자2개와 연산자를 함수에 넣고 결과를 res에 저장
        printf("%5.2lf %c %5.2lf = %7.2lf₩n", a, oper, b, res); // 연산결과를 출력한다.
    }

    fclose(fp); // 파일을 사용완료했으므로 파일포인터를 닫는다.
    return 0;
}
```

```c
double show(double a, char oper, double b)
{
    double lfResult = 0.0;

    if(oper == '+')
    {
        lfResult = a + b;
    }
    else if(oper == '-')
    {
        lfResult = a - b;
    }
    else if(oper == '*')
    {
        lfResult = a * b;
    }
    else if(oper == '/')
    {
        lfResult = a / b;
    }

    return lfResult;
}
```

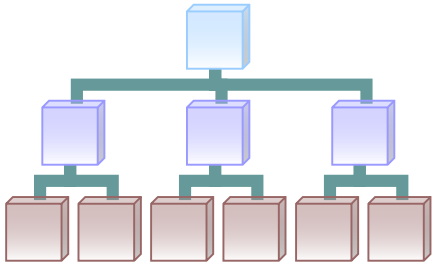
1장: 자료 구조와 알고리즘

- 목차
  - 1. 자료 구조와 알고리즘
  - 2. 데이터 타입, 추상 데이터 타입
  - 3. 알고리즘의 성능 분석
  - 4. 빅오 표기법

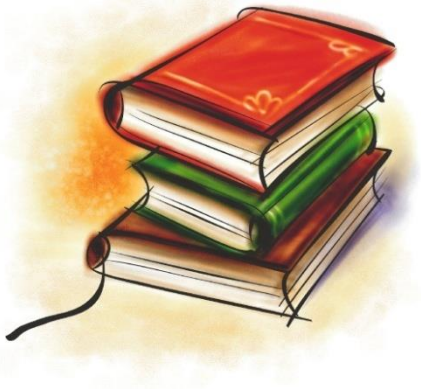
일상생활에서의 사물의 조직화



조직도

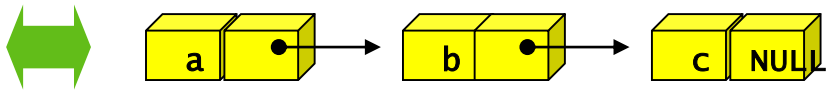


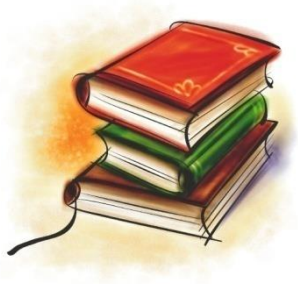
일상생활에서의
사물의 조직화

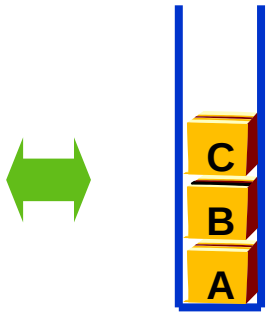


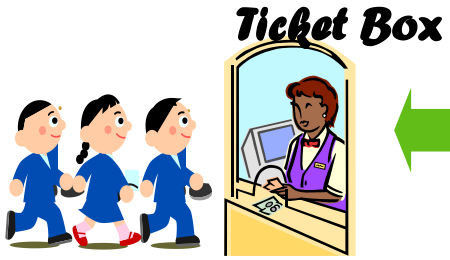
일상생활과 자료구조의 비교

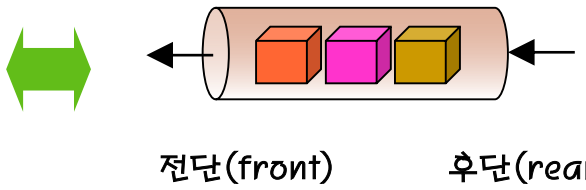
일상생활에서의 예	자료구조
물건을 쌓아두는 것	스택
영화관 매표소의 줄	큐
할일 리스트	리스트
영어사전	사전, 탐색구조
지도	그래프
조직도	트리











1. 자료구조와 알고리즘

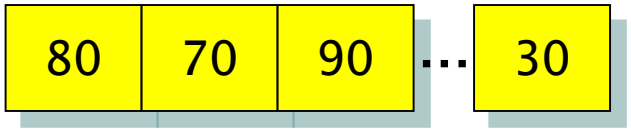
- 컴퓨터 프로그램은 자료를 처리하고 있고 이들 자료는 자료구조(Data Structure)를 이용하여 표현되고 저장된다. 또한 주어진 문제를 처리하는 절차가 필요한데 이것을 알고리즘(Algorithm)이라고 한다.
  - (예) 시험 성적을 읽어 최고 점수를 구하는 프로그램의 경우 성적을 배열에 저장**(자료구조)**하여 저장된 점수 중에 가장 큰 점수를 찾는 절차**(알고리즘)**가 필요하다.

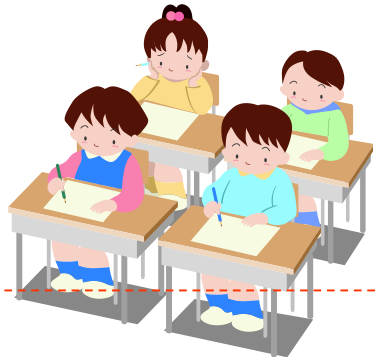
자료구조와 알고리즘

- 프로그램 = 자료구조 + 알고리즘
  - (예) 최대값 탐색 프로그램 = 배열+ 순차탐색

자료구조

score[]





알고리즘

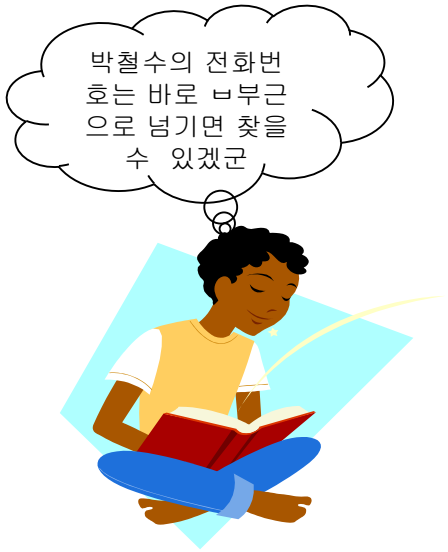
```
tmp←score[0];
for i ← 1 to n do
        if  score[i]>tmp
                then tmp←score[i];
```

자료구조와 알고리즘

- 자료구조와 알고리즘은 서로 밀접한 관계가 있음: 자료구조가 결정되면 그 자료구조에서 사용할 수 있는 알고리즘이 결정됨
- 컴퓨터가 복잡한 자료들을 빠르게 저장, 검색, 분석, 전송, 갱신하기 위해서는 자료구조가 효율적으로 조직화 되어야 함
- 따라서 각 응용 프로그램은 가장 적합한 자료구조와 알고리즘을 선택해야 함

알고리즘

- 컴퓨터를 이용하여 전화번호부에서 특정한 사람의 이름을 찾는 문제를 생각해 보자
  - 한가지 방법은 전화번호부의 첫 페이지부터 시작하여 한 장씩 넘기면서 특정한 사람을 찾는 방법으로 엄청난 시간이 소요됨
  - 다른 방법은 전화번호부의 이름들이 정렬되어 있음을 이용하는 방법임. “ㅂ”으로 시작하는 단어들이 있을만한 위치로 직접 가는 것임. 만약 너무 적게 갔으면 약간 앞으로 더 가고, 너무 많이 갔으면 조금 뒤로 가는 방법임
  - 이러한 방법들은 보통 프로그래밍 스타일이나 프로그래밍 언어 (C, Java 언어 등)과는 무관함
  - 알고리즘은 문제를 해결하는 방법을 정밀하게 장치가 이해할 수 있는 언어로 기술한 것임



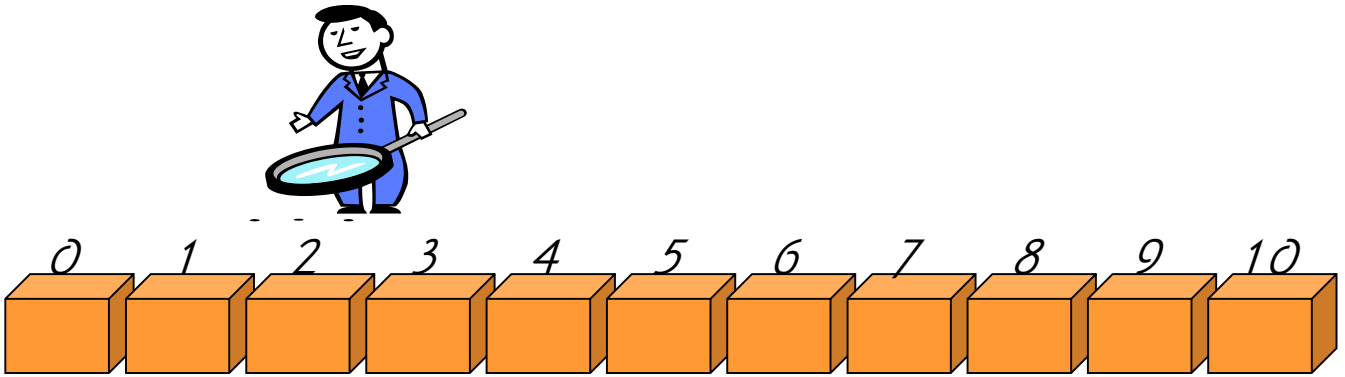
알고리즘

- 알고리즘(algorithm): 컴퓨터로 문제를 풀기 위한 단계적인 절차
- 모든 명령어들의 집합이 알고리즘이 되는 것이 아니라 알고리즘이 되기 위한 정의를 만족해야 함
- 알고리즘의 조건
  - 입 력 : 0개 이상의 입력이 존재하여야 한다.
  - 출 력 : 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: 자연어를 사용하기 때문에 모호성을 제거하기 위하여 명령어로 사용되는 단어들을 명백하게 정의해야 함
- 흐름도(flow chart): 알고리즘이 복잡해질수록 기술하기 힘들어짐
- 유사 코드(pseudo-code)나 C와 같은 프로그래밍 언어가 가장 많이 사용되고 있음
  - 유사 코드: 자연어보다 더 체계적이고 프로그래밍 언어보다는 덜 엄격한 언어로 알고리즘을 표현하는데 선호되는 표기법임

(예) 배열에서 최대값 찾기 알고리즘



자연어로 표기된 알고리즘

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

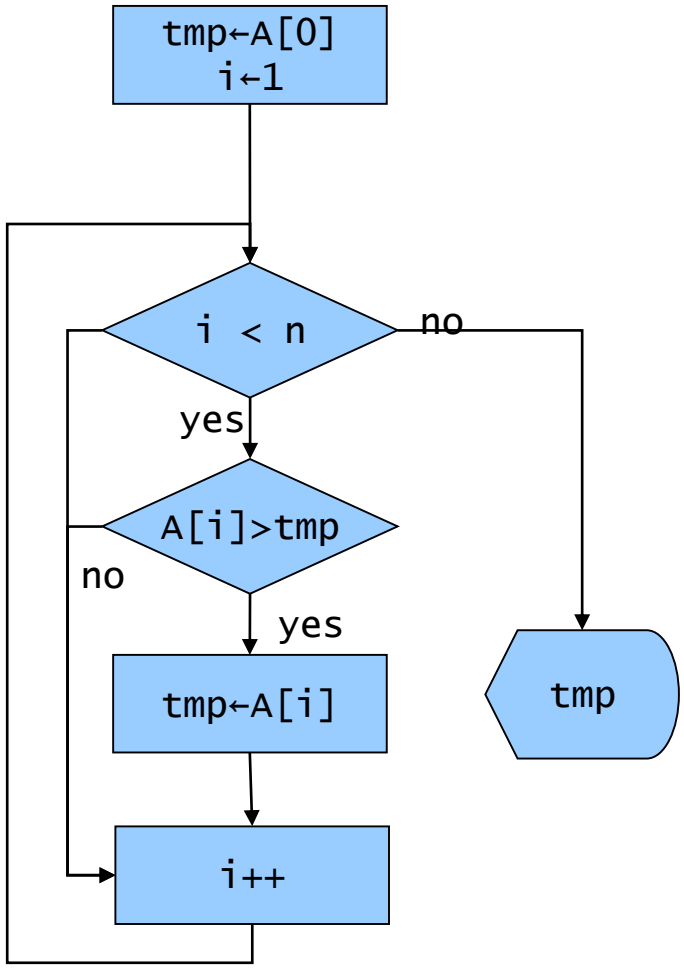
(예) 배열에서 최대값 찾기 알고리즘

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

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



유사코드로 표현된 알고리즘

- 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. 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

C로 표현된 알고리즘

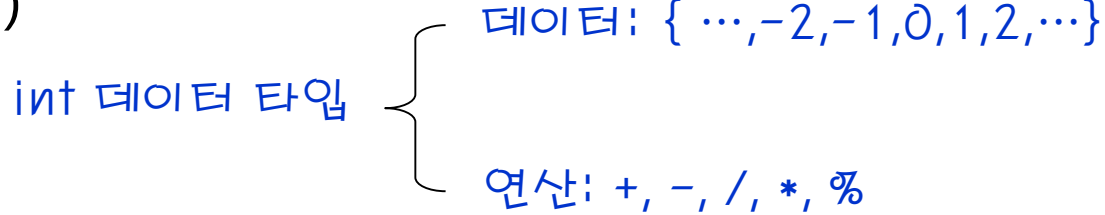
- 알고리즘의 가장 정확한 기술이 가능
- 반면 실제 구현시의 많은 구체적인 사항들이 알고리즘의 핵심적인 내용들의 이해를 방해할 수 있다.

```
#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;
}
```

2. 데이터 타입, 추상 데이터 타입

- 데이터 타입(data type)
  - 데이터의 집합과 연산의 집합

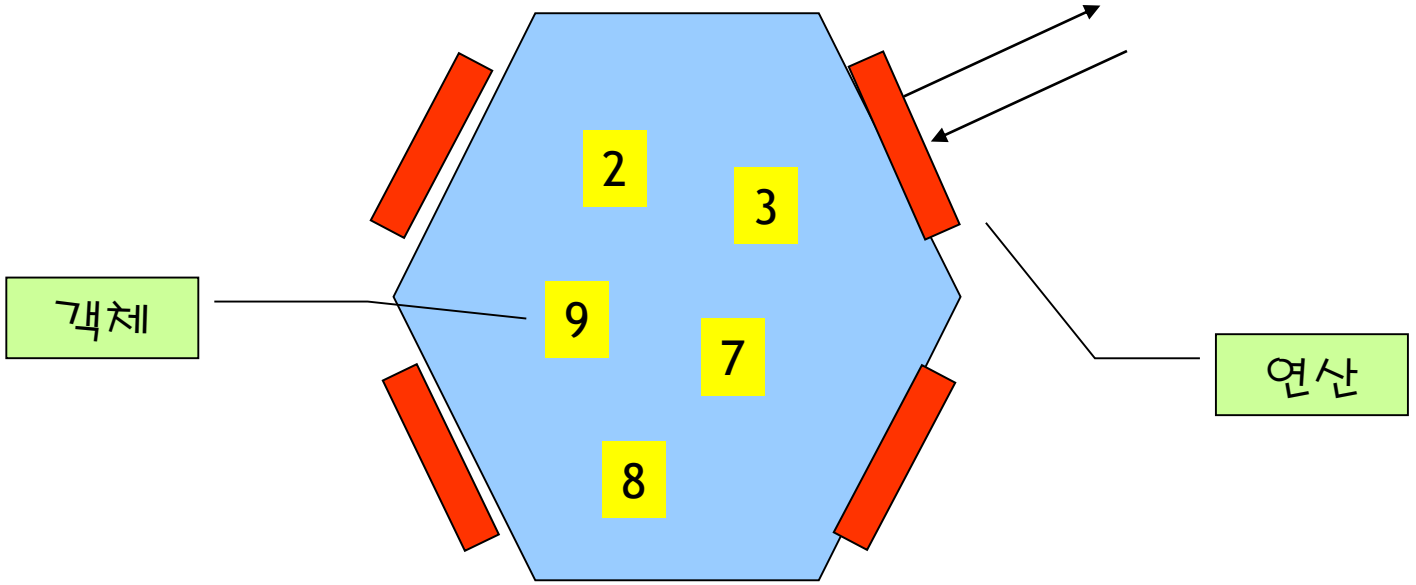
  (예)



- 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

- 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 타입

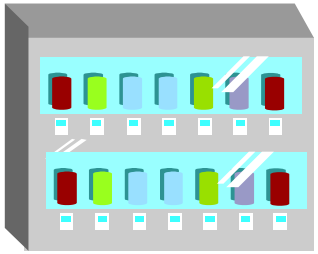
추상 데이터 타입

- 프로그램에서 **데이터**란 처리의 대상이 되는 모든 것을 말함
- 추상 데이터 타입이란 새로운 데이터 타입을 수학적으로 정의한 것으로 **자료 구조**는 이러한 추상 데이터 타입을 프로그래밍 언어로 구현한 것임
- 추상 데이터 타입은 추상 데이터 타입에 속하는 데이터 객체의 정의부터 시작되며 객체는 주로 집합의 개념을 사용하여 정의됨. 그 다음은 연산들이 정의 되며 연산의 정의에는 연산의 이름, 매개변수, 연산의 결과, 연산이 수행하는 기능의 기술 등이 포함됨
- 컴퓨터 프로그램 안에서 추상 데이터 타입이 구현될 때 보통 구현 세부사항은 외부에 알리지 않고 외부와의 인터페이스만을 공개함.사용자는 구현세부사항이 아닌 인턴페이스만을 사용하기 때문에 추상 데이터 타입의 구현은 후에 변경이 가능함: 전체 프로그램을 변경가능성이 있는 구현의 세부 사항으로부터 보호함 (**정보 은닉의 개념**)

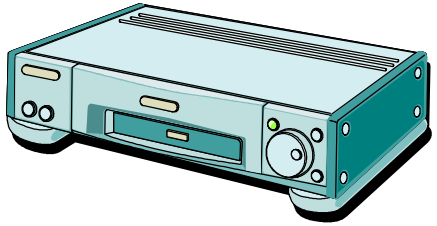
추상 데이터 타입의 예: 자연수

```
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 타입과 VCR



- 사용자들은 추상 데이터 타입이 제공하는 연산만을 사용할 수 있다.
- 사용자들은 추상 데이터 타입을 어떻게 사용하는지를 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.

추상 데이터 타입 이론

- 추상 데이터 타입의 이론은 Smalltalk나 C++, java와 같은 최근의 객체지향 프로그램 언어에 큰 영향을 줌
- 프로그래밍 언어에 따라서 추상 데이터 타입은 여러 방법으로 구현됨
  - C언어에서는 주로 구조체를 사용하여 구현되나 완벽하지는 않음
  - C++나 Java와 같은 객체지향언어에서는 클래스를 사용하여 추상 데이터 타입이 구현됨. 추상 데이터 타입의 객체는 클래스의 속성으로 구현되고 추상 데이터 타입의 연산은 클래스의 멤버 함수로 구현됨
  - 객체지향언어에서는 private나 protected 키워드를 이용하여 내부 자료의 접근을 제한할 수 있고 클래스는 상속으로 구성됨

3. 알고리즘의 성능 분석

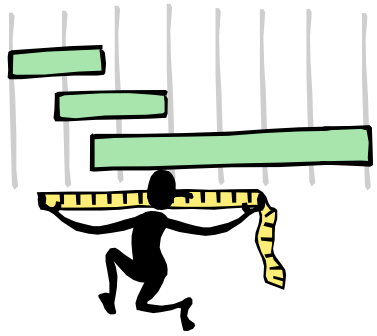
- 최근의 컴퓨터는 예전에 비해 엄청난 계산 속도와 방대한 메모리를 제공하고 있음. 하지만 최근에도 메모리를 효과적으로 사용하는 프로그램의 효율성은 여전히 중요함
  - 최근의 상용 프로그램은 처리해야 할 양이 많아지고 있음. 따라서 알고리즘의 효율성이 더욱 중요함

입력 자료의 개수	프로그램 A: n^2	프로그램 B: 2^n
n = 6	36초	64초
n = 100	10,000초	2^{100}초 = 4 X 10^{22}년

  - 사용자들은 여전히 빠른 프로그램을 선호하고 있음. 따라서 프로그래머는 하드웨어와는 상관없이 소프트웨어적으로 최선의 효율성을 갖는 프로그램을 제작해야 함. 일반적으로 수행 시간이 메모리 공간보다도 더 중요하게 고려되고 있음

알고리즘의 성능 분석 방법

- 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

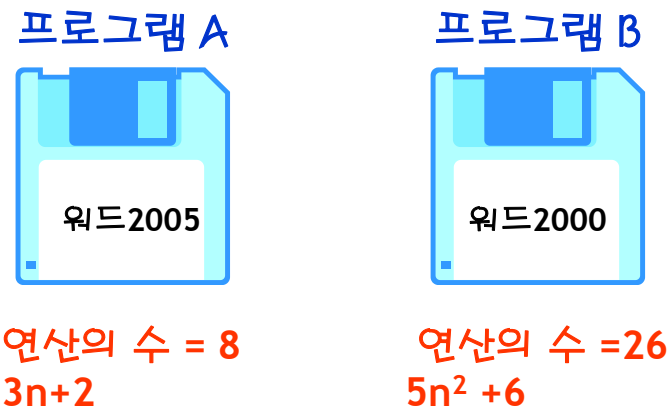
수행시간측정

- 컴퓨터에서 수행시간을 측정하는 방법에는 주로 clock 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 측정하고자 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 연산, 대입 연산, 비교 연산, 이동 연산의 기본적인 연산: 수행시간이 입력의 크기에 따라 변하면 안됨: 기본적인 연산만
- 알고리즘이 수행하는 연산의 개수를 계산하여 두 개의 알고리즘을 비교할 수 있다.
- 연산의 수행횟수는 고정된 숫자가 아니라 입력의 개수 n에 대한 함수->시간복잡도 함수라고 하고 T(n) 이라고 표기한다.



복잡도 분석의 예

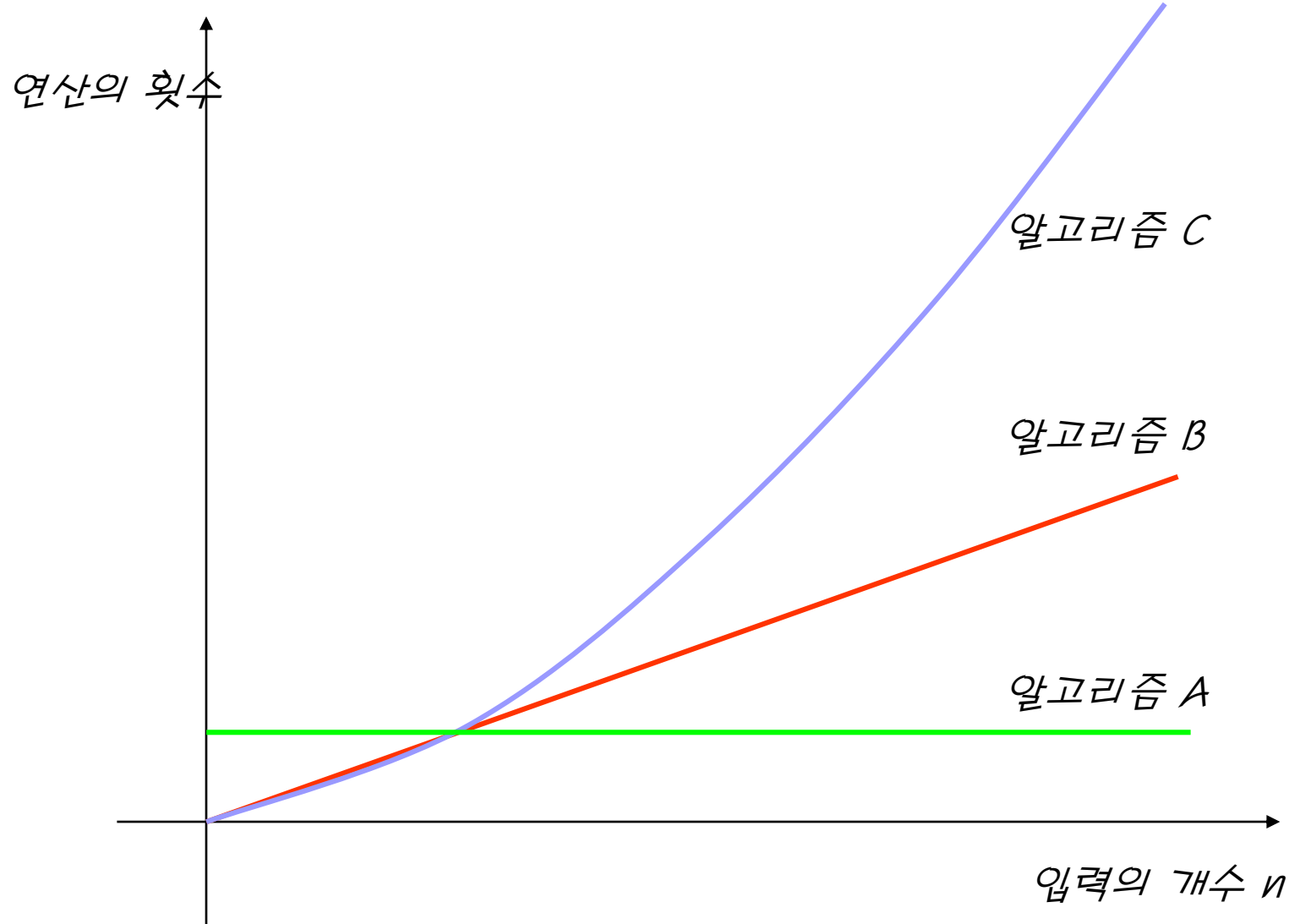
- n을 n번 더하는 문제:
  - 각 알고리즘이 수행하는 연산의 개수를 세어 본다.
  - 단 for 루프 제어 연산은 고려하지 않음.

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 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)
```

4. 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.
- 따라서 보통 시간복잡도 함수에서 가장 영향을 크게 미치는 항만을 고려하면 충분하다.

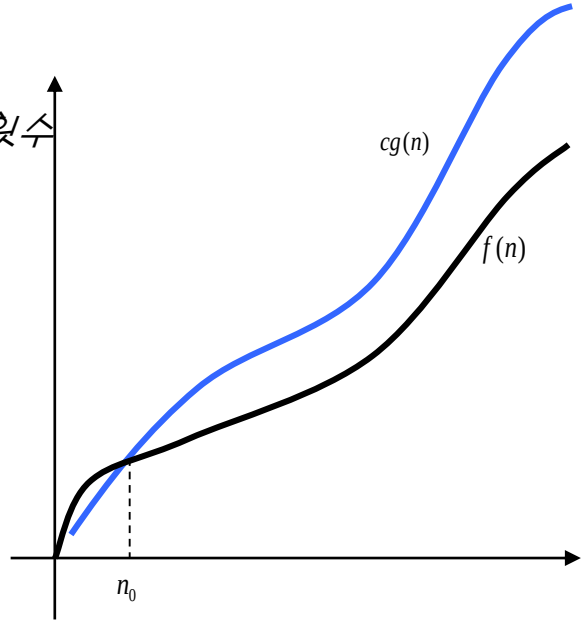
n=1000인 경우

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

99%

1%

- 빅오표기법: 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 c와 n_0가 존재하면 f(n)=O(g(n))이다.
- 빅오는 **함수의 상한을 표시**한다.
  - (예) f(n)=2n+1 이면O(n)이다. n_0 =2, c=3일때 $n \geq 2$에 대하여 $2n+1 \leq 3n$ 이므로 2n+1 = O(n)



빅오 표기법의 예

예제 1.1 빅오 표기법

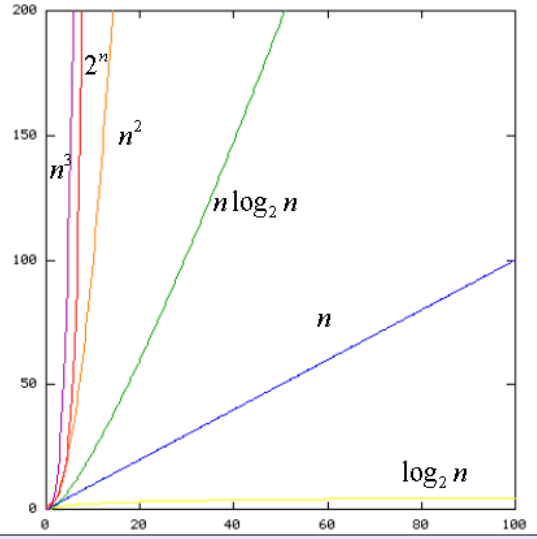
- $f(n)=5$이면 $O(1)$이다. 왜냐하면 $n_0=1$, $c=10$일 때, $n \geq 1$에 대하여 $5 \leq 10 \cdot 1$이 되기 때문이다.
- $f(n)=2n+1$이면 $O(n)$이다. 왜냐하면 $n_0=2$, $c=3$일 때, $n \geq 2$에 대하여 $2n+1 \leq 3n$이 되기 때문이다.
- $f(n)=3n^2+100$이면 $O(n^2)$이다. 왜냐하면 $n_0=100$, $c=5$일 때, $n \geq 100$에 대하여 $3n^2+100 \leq 5n^2$이 되기 때문이다.
- $f(n)=5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

빅오 표기법

- 빅오 표기법은 시간 복잡도 함수의 증가에 별로 기여하지 못하는 항을 생략함으로써 시간 복잡도를 간단하게 표현
- 빅오 표기법을 얻는 간단한 방법
  - 기본 연산 횟수가 다항식으로 표현되었을 경우 다항식의 최고차항만을 남기고 다른 항들과 상수항을 버림. 최고차항의 계수도 버리고 최고차항의 차수만을 사용함.
  - ex) $a_m n^m + \cdots\cdots + a_1 n + a_0 = O(n^m)$
  - $7n-3 = O(n)$
  - $8n^2\log n + 5n^2 + n = O(n^2\log n)$

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

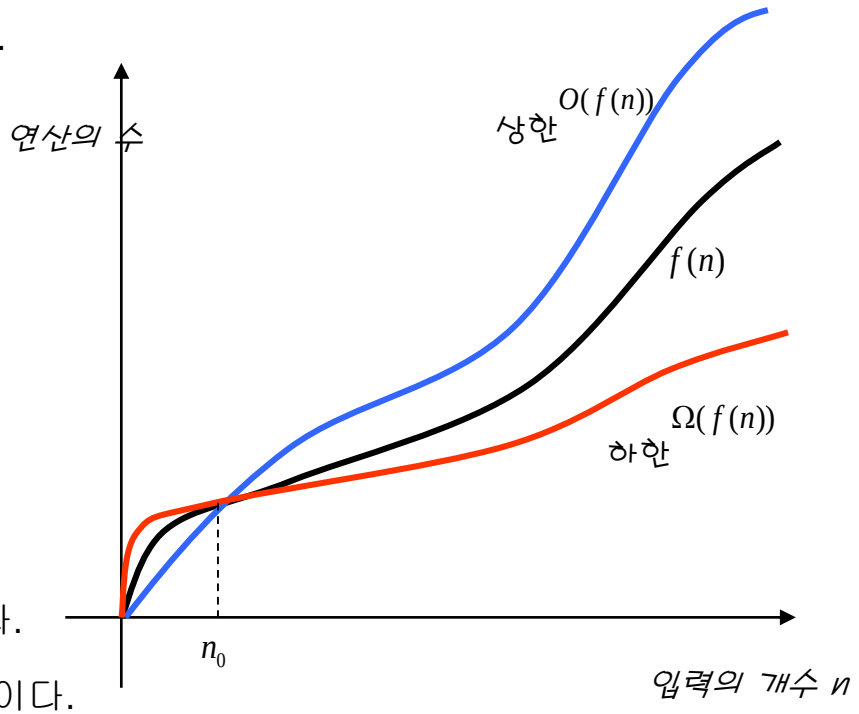


시간복잡도	n					
	1	2	4	8	16	32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40326	20922789888000	26313×10^{33}

빅오 표기법이외의 표기법

- 빅오메가 표기법
  - 모든 $n \geq n_0$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 표시한다.
  - (예) $n \geq 5$ 이면 $2n+1 < 10n$ 이므로 $n = \Omega(n)$

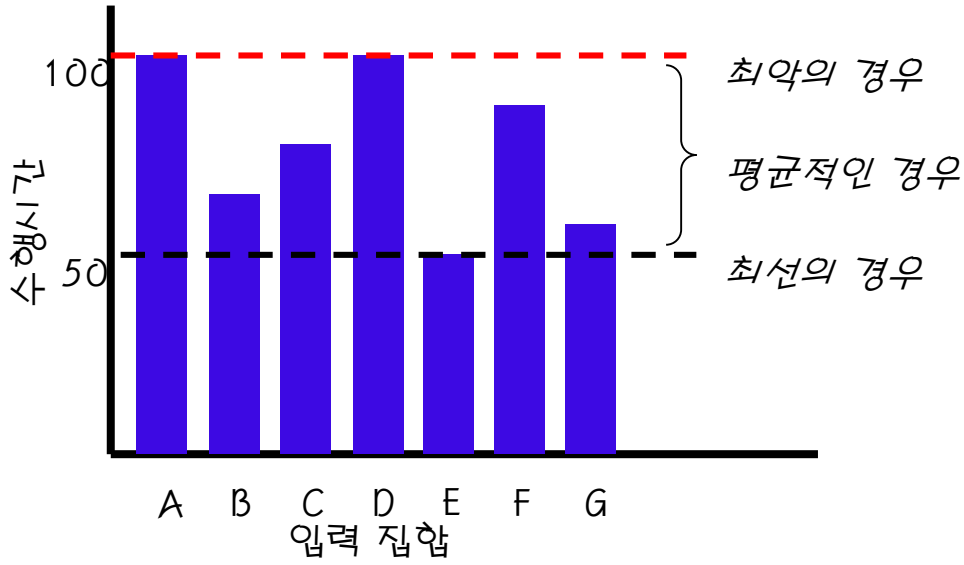
- 빅세타 표기법
  - 모든 $n \geq n_0$에 대하여 $c1|g(n)| \leq |f(n)| \leq c2|g(n)|$을 만족하는 3개의 상수 c1, c2와 n_0가 존재하면 $f(n)=\theta(g(n))$이다.
  - 빅세타는 함수의 하한인 동시에 상한을 표시한다.
  - $f(n)=O(g(n))$이면서 $f(n)= \Omega(g(n))$이면 $f(n)= \theta(n)$이다.
  - (예) $n \geq 1$이면 $n \leq 2n+1 \leq 3n$이므로 $2n+1 = \theta(n)$



최선, 평균, 최악의 경우

- 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
- 평균의 경우(average case): 수행시간이 평균적인 경우
- 최악의 경우(worst case): 수행 시간이 가장 늦은 경우



- 최선의 경우: 의미가 없는 경우가 많다.
- 평균적인 경우: 계산하기가 상당히 어려움.
- 최악의 경우: 가장 널리 사용된다. 계산하기 쉽고 응용에 따라서 중요한 의미를 가질 수도 있다.
- (예) 비행기 관제업무, 게임, 로보틱스

최선, 평균, 최악의 경우

- (예) 순차탐색
- 최선의 경우: 찾고자 하는 숫자가 맨앞에 있는 경우

  ∴ O(1)

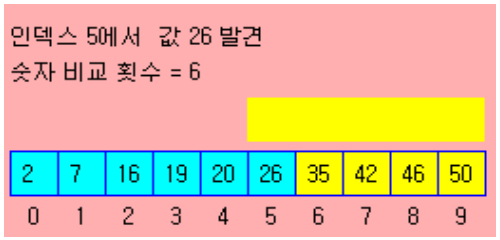
- 최악의 경우: 찾고자 하는 숫자가 맨뒤에 있는 경우

  ∴ O(n)

- 평균적인 경우: 각 요소들이 균일하게 탐색된다고 가정하면

  (1+2+…+n)/n=(n+1)/2

  ∴ O(n)



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

- 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

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

  (예) typedef int element;

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

typedef <새로운 타입의 정의> <새로운 타입 이름>;