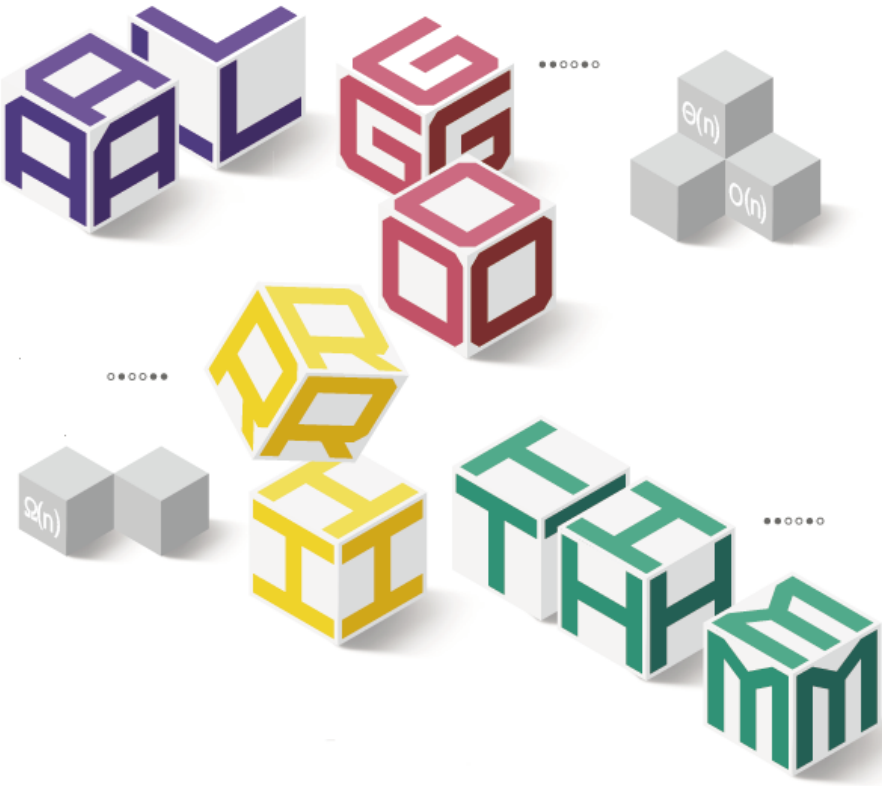


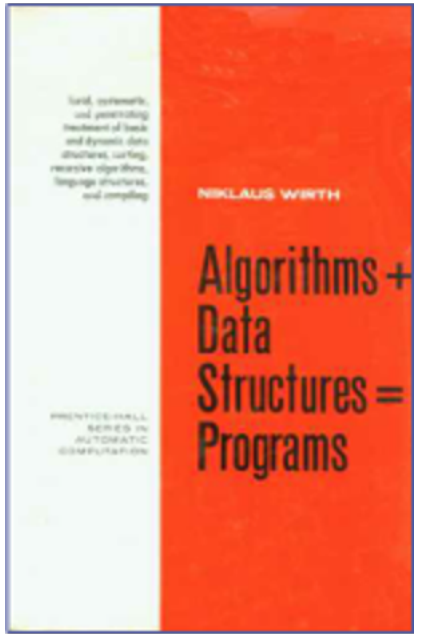
관계 중심의 사고법

쉽게 배우는 알고리즘

수업 소개

강의 개요

- 과목 개요
  - 프로그램을 보다 효율적으로, 보다 지능적으로 작성하기 위하여 필요한 과목
  - 새로운 방법이나 이론을 알고리즘으로 기술함
- 알고리즘의 응용 분야
  - 모든 프로그램의 설계
  - 문제 및 해결 방법의 정형적 표현
  - 프로그램의 효율성 및 복잡도 분석
- Algorithms + Data Structures = Programs



강의 개요

- 선행과목
  - 자료구조론
    - 필수사항으로 선수내용(연결리스트, 스택, 큐, 트리) 이해
    - 해당 내용은 수업에서 다루지 않음
  - 프로그래밍
    - C 언어(C++, Java, C#, Python)
      - 함수, 포인터, 구조체
    - 프로그래밍 리포트(수업의 주요 알고리즘의 적용을 다룸)

강의 개요

- 자료구조는 빌딩을 지을 때 재료의 역할을 함
- 자료구조에 따른 정렬, 검색 및 자료 다루기를 위한 알고리즘을 사용함
- 신중히 선택한 자료구조는 보다 효율적인 알고리즘을 사용할 수 있게 함

알고리즘 활용

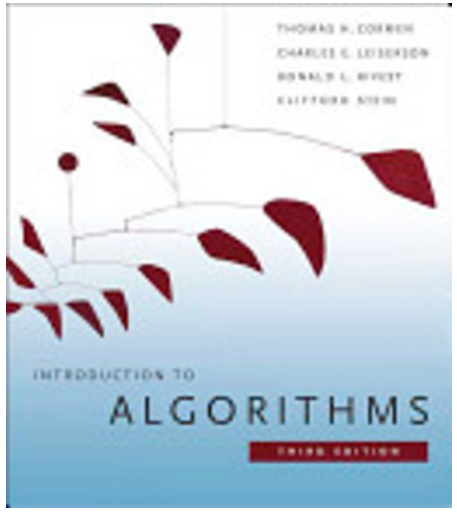
https://visualgo.net/en

강의 개요

- 교재
  - IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법
  - 저　　자 : 문병로
  - 출 판 사 : 한빛미디어(주)

강의 개요

- 부교재
  - Introduction to Algorithms, 3rd, MIT Press, 2009
  - Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein



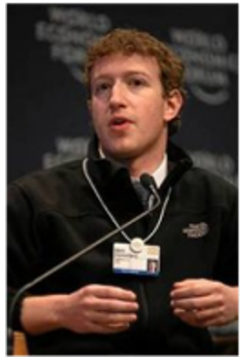
알고리즘 이란

- 알고리즘(algorithm, AL-go-rith-um) 은 문제 해결을 위한 절차(procedure) 또는 공식(formula)임
- 계산을 위한 단계별 절차(step-by-step procedure)
- 문제 해결을 위한 모호하지 않은 명령어의 순서(sequence of unambiguous instructions)

알고리즘 이란

Sergey Brin

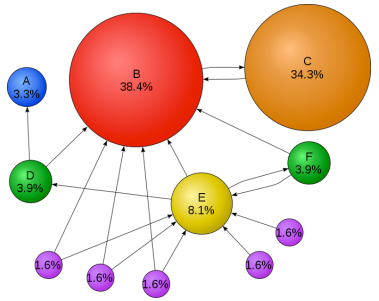
Lawrence Page

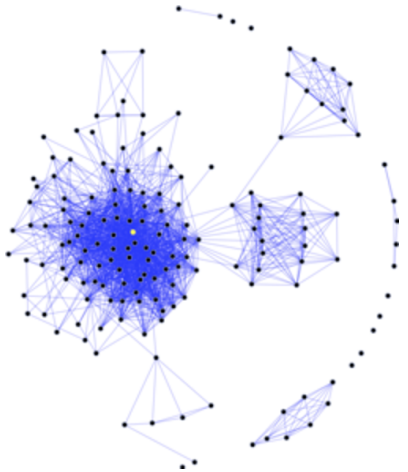


Mark Zuckerberg

Google과 Facebook의 창립자는 무엇을 했는가

- Google의 성공 원인
  - PageRank 알고리즘 (+자료구조)
- Facebook의 성공 원인
  - 효율적인 인적 네트워크 관리 알고리즘 (+자료구조)





- 이들의 공통점은 필요한 알고리즘을 자신들의 시스템에 매우 효율적으로 적용했다는 점!!!

여러분이 알고리즘을 배워야 하는 이유는 ?

- 개발 언어 또는 플랫폼에 독립적으로 문제해결 방법을 익히기 위해
- (개발자,취업)/(연구자, 진학) 로서 주어진 문제의 효과적인 해결방안을 찾기 위해
- 국내외 대기업 및 중견기업의 면접에 대비하기 위해
- 정보처리기사 시험 과목이기 때문에

다루는 내용

도입	알고리즘 학습을 위한 기본 지식	1장 알고리즘이란 2장 알고리즘 설계와 분석의 기초 3장 점화식과 알고리즘 복잡도 분석
전통적 문제	특정 문제를 통한 사고법 훈련	4장 정렬 5장 선택 알고리즘 12장 문자열 매칭
자료의 저장과 검색	알고리즘의 중요한 응용 분야의 하나인 자료의 저장 및 검색을 위한 자료구조와 알고리즘	6장 검색 트리 7장 해시 테이블
대상의 표현과 해결	특정 대상(집합, 그래프)의 표현과 문제 해결법	8장 집합의 처리 10장 그래프
구조와 문제	특이한 추상적 구조를 가진 문제들의 해결법	9장 동적 프로그래밍 11장 그리디 알고리즘
계산의 한계	문제 해결이 항상 빠른 시간에 가능하지 않다는 사실	13장 NP-완비
문제 해결의 다른 시각	상태 공간에서의 이동으로 해를 찾는 과정 파악	14장 상태 공간 트리의 탐색

주차별 계획

주	해당 장	주제
1	1장	알고리즘이란
2	2장	알고리즘 설계와 분석의 기초(2.3 점근적 표기의 엄밀한 정의는 선택적)
3	2장 3장	점화식과 점근적 복잡도 분석
4	4장	정렬 (기본정렬: 선택, 버블, 삽입 정렬알고리즘)
5	4장	정렬 (고급정렬: 병합, 퀵 정렬알고리즘, 힙정렬알고리즘)
6	4장 5장	정렬 (고급정렬: 힙정렬 알고리즘 분석)/선택알고리즘
7	수시고사	
8	6장	검색 트리 (트리표현)
9	6장	검색트리(BST, 6.5 다차원 트리 개념)
9	7장	해시 테이블 (해쉬테이블 소개:활용, 해쉬함수 이론)
10	7장	해시 테이블 (충돌해결)
11	9장	동적프로그래밍 (개념이해)
14	10장	그래프 (표현), 그래프 (DFS, BFS)
15	기말 평가	

퀴즈 및 과제 관련

- 주차 별 플립러닝에 따른 퀴즈 (성적반영)
- 프로그래밍 과제 (4회, 성적반영)
- 과제 작성 유의사항
  - 일반적으로 7일 기한
  - 구글클래스룸에 기한이 제시됨
  - 과제 제출기한 지연시 감점
  - (중요) 제공된 코드(주어진 조건)을 사용하여 프로그래밍하여야함
  - (중요)과정에서 대한 설명기술

평가방법 및 평가기준

- A : 알고리즘을 이해하고 이를 프로그램한다
- B : 알고리즘을 이해하고 있으며 일부 프로그램 한다
- C : 알고리즘을 일부 이해하고 있음, 그러나 프로그램 하지 못한다
- D/F : 알고리즘을 이해하지 못함

1장. 알고리즘이란

알고리즘은 문제 해결 과정을 묘사하는 것

- 문제 해결 절차를 체계적으로 기술한 것
- 문제의 요구 조건
  - 입력과 출력으로 명시할 수 있다
  - 알고리즘은 입력으로 부터 출력을 만드는 과정을 기술

입출력의 예

- 문제
  - 100명의 학생의 시험점수의 최대값을 찾으라
- 입력
  - 100명의 학생들의 시험점수
- 출력
  - 위 100개의 시험점수들 중 최대값

바람직한 알고리즘

- 명확해야 한다
  - 이해하기 쉽고 가능하면 간명하도록
  - 지나친 기호적 표현은 오히려 명확성을 떨어뜨림
  - 명확성을 해치지 않으면 일반언어의 사용도 무방
- 효율적이어야 한다
  - 같은 문제를 해결하는 알고리즘은 여러개가 있을 수 있으며
  - 이들의 수행 시간은 수백만 배 이상 차이날 수 있다

알고리즘 작성 예

- 알고리즘 LargestNumber
- Input: A list of numbers L.
- Output: The largest number in the list L.

```
largest ← L[0]
for each item in L, do
    if item > largest
        largest ← item
return largest
```

수행시간측정

- 컴퓨터에서 수행시간을 측정하는 방법 clock 함수가 사용

clock_t clock(void);

```
#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);
}
```

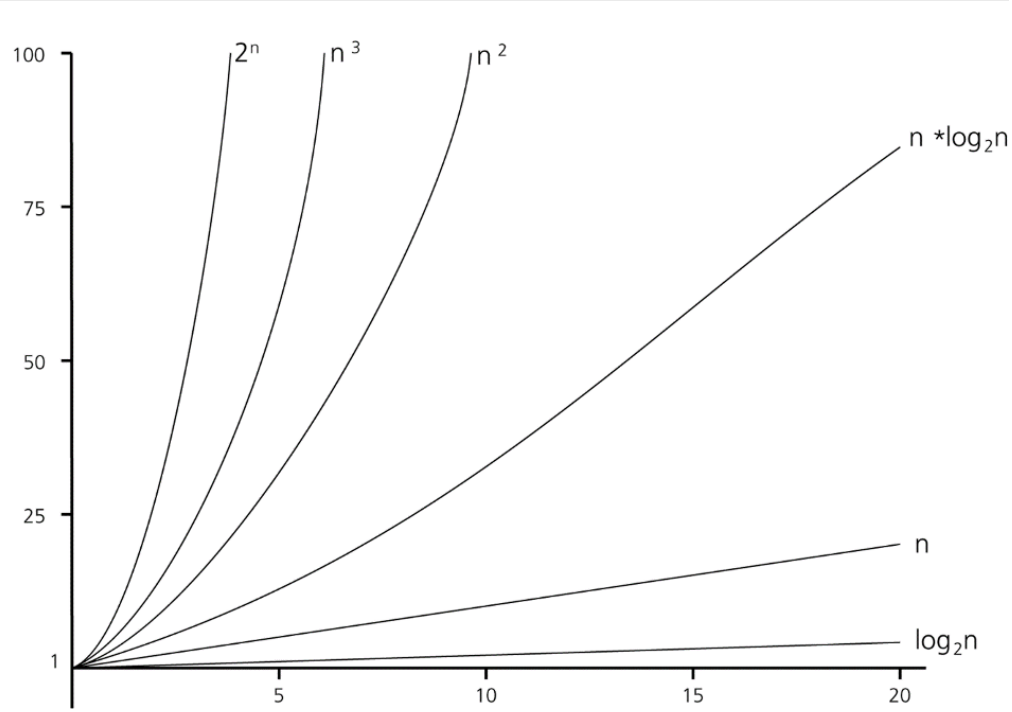
- 프로그램을 사용하여 알고리즘의 성능을 평가할 때 문제점은 무엇인가

수행시간측정

- 수행시간 측정 방법의 문제점
  - 프로그램을 해야 함
  - 성능 측정이 어려운 경우, 블랙박스로 소스 공개가 안되는 경우 알고리즘 평가가 쉽지 않음
  - 두 개 알고리즘 비교 시 공정한 평가를 위해 동등한 조건을 만들어야 함 (동일 컴퓨터, 동일 OS, 동일한 시스템 상황)

알고리즘의 수행 시간

- 함수의 x는 자료의 크기 y는 수행시간을 표현함



알고리즘의 수행 시간

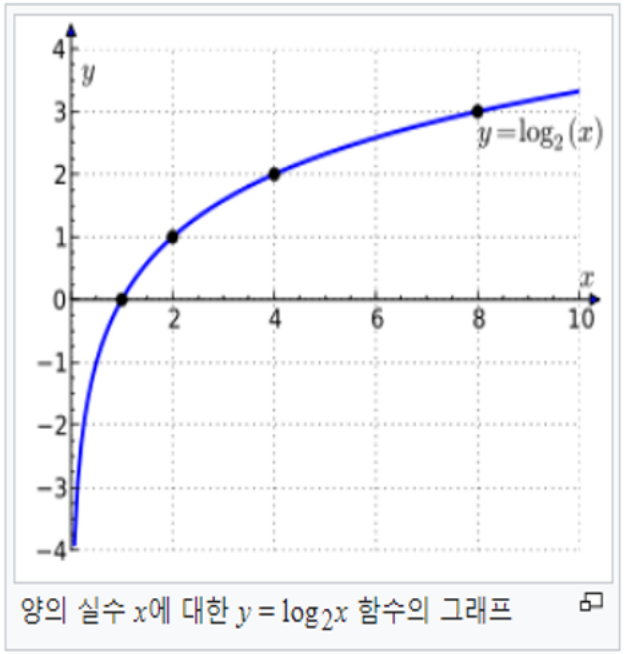
이진 로그

위키백과, 우리 모두의 백과사전.

수학에서 **이진 로그** (binary logarithm)는 밑이 2인 로그이다. $\log_2$ 또는 lb[1]로 표기하며, 2의 거듭제곱의 역함수이다. 양의 실수 n과 실수 x에 대하여 $x = \log_2 n$은 $2^x = n$이라는 것과 같다.

$$x = \log_2 n \iff 2^x = n.$$

예를 들어 $\log_2 1 = 0$, $\log_2 2 = 1$, $\log_2 4 = 2$, $\log_2 32 = 5$이다.



양의 실수 x에 대한 $y = \log_2 x$ 함수의 그래프

각주 [편집]

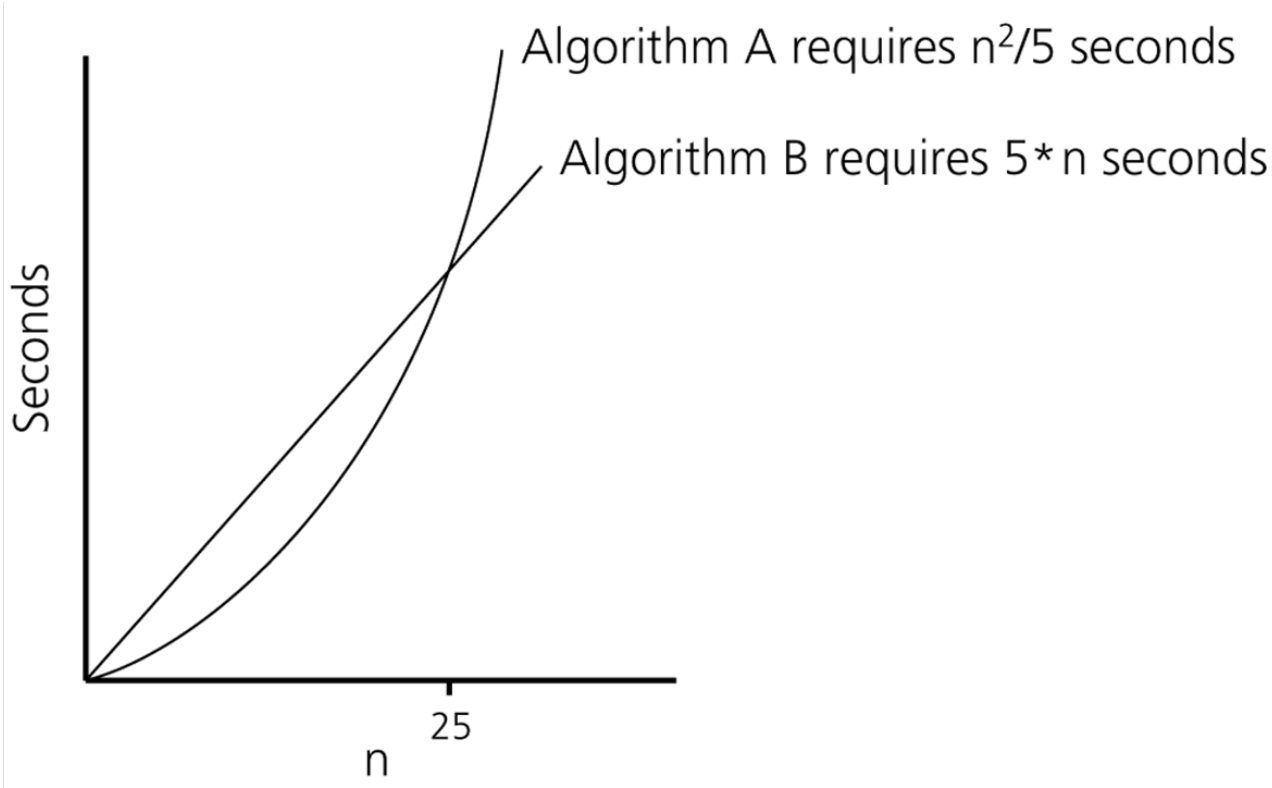
1. ↑ ISO 31-11과 ISO 80000-2

같이 보기 [편집]

- 로그
- 자연로그
- 상용로그

알고리즘의 수행 시간

- 함수의 x는 자료의 크기 y는 수행시간을 표현함



알고리즘의 수행 시간

	n					
Function	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

알고리즘의 분석

- 크기가 작은 문제
  - 알고리즘의 효율성이 중요하지 않다
  - 비효율적인 알고리즘도 무방
- 크기가 충분히 큰 문제
  - 알고리즘의 효율성이 중요하다
  - 비효율적인 알고리즘은 치명적
- 입력의 크기가 충분히 큰 경우에 대한 분석을 점근적 분석이라 한다
  - 데이터의 개수 $n \to \infty$ 일때 수행시간이 증가하는 growth rate로 시간복잡도를 표현하는 기법

알고리즘의 분석

- 알고리즘의 수행 시간을 좌우하는 기준은 다양하게 잡을 수 있다
  - 예: for 루프의 반복횟수, 특정한 행이 수행되는 횟수, 함수의 호출횟수, ...
  - 몇 가지 간단한 경우의 예를 통해 알고리즘의 수행 시간을 살펴본다

알고리즘의 분석

sample1(A[ ], n)

{

k = $\lfloor n/2 \rfloor$;

return A[k];

}

- n에 관계없이 상수 시간이 소요된다

알고리즘의 분석

- 가장 자주 실행되는 문장이 n번이라면 모든 문장의 실행 횟수의 합은 n에 선형적으로 비례함

```
int sum(A[], int n)
{
   sum ← 0 ;
   for i ← 1 to n
      sum=sum+A[i];
   return sum;
 }
```

- 선형 시간복잡도를 가진다고 말하고 $O(n)$이라고 표기한다

알고리즘의 분석

```
sample3(A[ ], n)
{
        sum ← 0 ;
        for i ← 1 to n
                for j ← 1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

- n^2에 비례하는 시간이 소요된다

알고리즘의 분석

```
sample4(A[ ], n)
{
    sum ← 0 ;
    for i ← 1 to n
        for j ← 1 to n {
            k ← A[1 ... n]에서 임의로 ⌊n/2⌋개를 뽑을 때 이들 중 최댓값 ;
            sum ← sum + k ;
        }
    return sum ;
}
```

- n^3에 비례하는 시간이 소요된다

알고리즘의 분석

```
sample5(A[ ], n)
{
        sum ← 0 ;
        for  i ← 1 to n-1
                for  j ←  i+1 to n
                        sum← sum+ A[i]*A[j] ;
        return sum ;
}
```

- n^2에 비례하는 시간이 소요된다

알고리즘의 분석

```
factorial(n)
{
        if (n=1) return 1 ;
        return n*factorial(n-1) ;
}
```

- n에 비례하는 시간이 소요된다

재귀와 귀납적 사고

- 재귀=자기호출(recursion)
- 재귀적 구조
  - 어떤 문제 안에 크기만 다를 뿐 성격이 똑같은 작은 문제(들)가 포함되어 있는 것
  - 예1: factorial
    - N! = N×(N-1)!
  - 예2: 수열의 점화식
    - $a_n = a_{n-1} + 2$

1부터 n까지 연속한 숫자의 합

같은 문제를 해결하는 서로 다른 문제해결 방법

두개의 방법

```
def sum_A(n):
    sum = 0
    for i in range(1, n+1):
        sum += i
    return sum
```

필요한 계산횟수가 입력크기 n과 비례함

```
def sum_B(n):
    sum = n * (n + 1) // 2
    return sum
```

필요한 계산횟수가 입력크기 n과 무관함

알고리즘 분석

- 입력 크기와 계산 횟수
  - 10000 까지 합

  sum (A) ran: 0.8322909750004328 milliseconds

  sum (B) ran: 0.00029702899882977363 milliseconds

  - 1000000 까지 합

  sum (A) ran: 91.97638053299852 milliseconds

  sum (B) ran: 0.00033955800063267816 milliseconds

- 계산 복잡도 표현을 위해 빅오 표기를 사용함

알고리즘 분석

- 재귀호출을 사용하여 1에서 N 까지 합구하기

```
def sumN(n) -> int:
    if n == 1:
        return 1
    else:
        return n + sumN(n-1)

n = 10
result = sumN(n)
print (f"sum of {n} = {result}")
```

문제정의

- 주어진 리스트 안의 자료를 작은 수부터 큰 수 순서로 배열하고자 함

- 입력
  - 배열, 리스트 (순서가 맞지 않은)
    [35, 9, 2, 85,17]

- 출력
  - 순서대로 정렬된 리스트
    [2, 9, 17, 35, 85]

분석

- 자료를 크기 순서로 배치하기 위해 두 자료를 비교함
- 비교 횟수에 의해 계산 복잡도가 결정됨
- 총 비교 횟수는 n(n-1)/2 임