



1장. 기본 개념



목차

- System Life Cycle의 개념
- 알고리즘 명세(Algorithm Specification)
- 데이터 추상화(Data Abstraction)
- 성능 분석(Performance Analysis)
- 성능 측정(Performance Measurement)



1. System Life Cycle의 개념

- 요구사항 분석 단계(Requirements)
 - 문제와 결과 정보를 규정
- 시스템 분석 단계(Analysis)
 - 상향식 설계(Bottom-up Analysis)
 - 세부 기능의 구현을 초기에 강조
 - 결과 프로그램 = 세부 기능들의 조합
 - 단점: 주어진 문제의 고유 특성을 고려하지 않는 설계가 될 위험이 있음.
 - 하향식 설계(Top-down Analysis)
 - 프로그램을 관리 가능한 세그먼트들로 분할
 - 단점: "도미노 효과"가 발생 가능
사용자의 요구가 달라질 수 있음



System Life Cycle의 개념(2)

- 설계 단계(Design)
 - 데이터와 연산의 관점에서 접근
 - 데이터 관점: 추상적 데이터 타입(Abstract Data Type)
 - 연산의 관점: 알고리즘(Algorithm)
 - 프로그래밍 언어에 독립적
- 구현 단계(Refinement and Coding)
 - 데이터의 구현 방법을 선택
 - 데이터에 대한 각 연산들의 구현 알고리즘을 선택
 - 구현 단계의 주요 고려 사항: 성능(Performance)

System Life Cycle의 개념(3)

■ 검증 단계(Verification)

■ 정확성 증명(Correctness Proof)

- 수학적 기법: 시간이 많이 소요. 경우에 따라 불가능
- 정확하다고 알려져 있는 알고리즘을 사용
- 구현 단계 이전이나 구현 중에 진행 가능

■ 검사(Testing) 개발자가 Testing 안전하게 됨

- 실행 코드와 테스트 데이터가 요구됨
- 양질의 테스트 데이터는 실행 코드의 모든 부분을 검사
- 프로그램의 실행 시간도 측정

■ 오류 제거(Error Removal)

- 경우에 따라 가장 많은 시간이 소요될 수 있음
- 문서화가 되어 있지 않으면서, 뒤죽박죽 섞여있는 코드에서 오류를 제거:
Programmer's nightmare!

■ 좋은 프로그램이란?

- 문서화 (주석, 변수, 함수 이름)
- 전체 프로그램이 기능적으로 깔끔하게 분리
- 각 부분은 인자 전달(메시지 전달, 함수 호출 등)로 상호 동작



2. 알고리즘 명세

- **알고리즘의 정의**
 - 어떤 일을 수행하기 위한 유한 개의 명령어들의 나열.
- **모든 알고리즘들이 만족해야 할 조건들**
 - **입력(Input)**: 0 혹은 그 이상의 입력이 존재
 - **출력(Output)**: 적어도 하나 이상의 결과물이 출력
 - **명확성(Definiteness)**: 알고리즘을 구성하는 명령어들의 의미는 명확하여야 하며, 애매모호해서는 안 된다.
 - **유한성(Finiteness)**: 알고리즘은 한정된 수의 명령어들을 수행한 후 종료하여야 한다.
 - **실행가능성(Effectiveness)**: 모든 명령어들은 실행 가능하여야 한다.



알고리즘의 예

- 코끼리를 냉장고에 넣는 방법 알고리즘 X
(입력: 냉장고와 코끼리, 출력: 코끼리가 들어간 냉장고)
 1. 냉장고 문을 연다.
 2. 코끼리를 냉장고에 넣는다. ← 실행가능성 X
 3. 냉장고 문을 닫는다.
- 라면을 끓이는 법 알고리즘 O
(입력: 라면 재료, 출력: 맛있게 끓인 라면)
 1. 냄비에 물을 500ml 넣고 거품이 날 때까지 끓인다.
 2. 라면과 수프를 함께 넣는다.
 3. 거품이 나면 불을 끈다.



알고리즘의 예: Selection Sorting(1)

- 문제
 - $n \geq 1$ 개의 정수를 정렬하는 프로그램을 작성
- 단순한 해결 방법
 - 현재까지 정렬되지 않은 정수들 중에서 가장 작은 것을 찾아 정렬 리스트에 추가하자.
 - 알고리즘이 아니다. 왜?
 - 애매모호한 명령어를 포함
 - 초기에 정수들을 어디에, 그리고 어떻게 저장할 것인가? 에 대한 설명이 없음.
 - 뿐만 아니라, 정렬 리스트를 어떻게 구성할 지에 대한 설명도 없음.

Selection Sorting(2)

- 보다 구체적인 해결 방법

```
for (i = 0; i < n; i++) {  
    Examine list[i] to list[n-1];  
    Suppose that the smallest integer is at list[min];  
    Interchange list[i] and list[min];  
}
```

- Interchange의 구현 방법

로딩, 실행 ^

- swap(&a, &b) \Leftarrow Program 1.2 함수로 jump
- #define SWAP(x, y, t) ((t) = (x), (x) = (y), (y) = (t)) 소스코드를 바꾸고 컴파일
대신 소스코드가 길어짐

- 매크로와 함수

- 매크로: 효율적(실행 시간)
- 함수: 디버깅 용이(실행 코드의 크기)



Program 1.3: Swap 함수

```
void swap(int *x, int *y)
{
    int temp = *x;

    *x = *y;
    *y = temp;
}
```

swap(&a, &b)

Program 1.4: Selection Sorting 함수(1)

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 101
#define SWAP(x,y,t) ((t)=(x), (x)=(y), (y)=(t)) // 매크로로 구현
void sort(int [ ], int); // selection sort
void main(void)
{
    int i, n, list[MAX_SIZE];
    printf("Enter the number of numbers to generate: ");
    scanf("%d", &n);
    if (n<1 || n>MAX_SIZE) { // error 처리
        fprintf(stderr, "Improper value of n\n");
        exit(1);
    }
    for (i=0; i<n; i++) { // n개의 정수를 random하게 생성
        list[i] = rand() % 1000;
        printf("%d ", list[i]);
    }
}
```

Program 1.4: Selection Sorting 함수(2)

```
sort(list, n);           // sort 함수를 호출. 인자는 배열과 정수의 개수
printf("\n Sorted array:\n ");
for (i = 0; i < n; i++)
    printf("%d ", list[i]);    // 정렬된 정수를 출력
printf("\n");
}
```

```
void sort(int list[], int n)
{
```

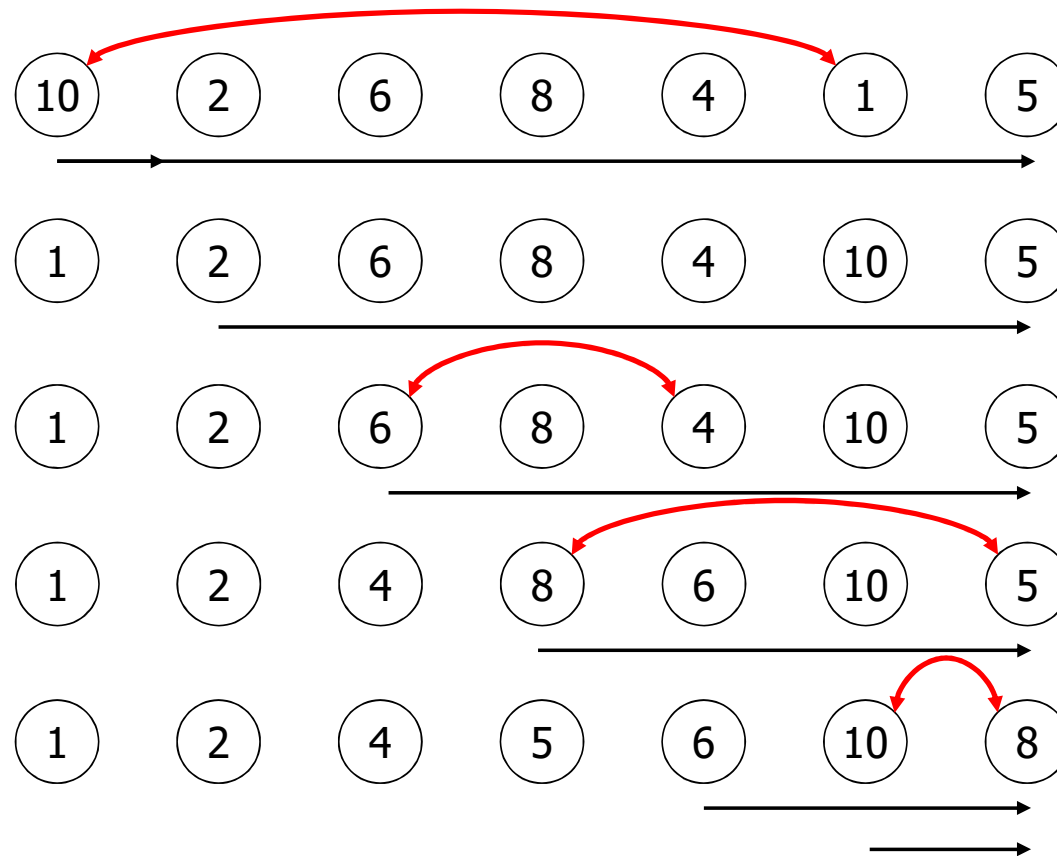
```
    int i, j, min, temp;
    for (i = 0; i < n-1; i++) {
        min = i;    n-2까지
        for (j = i+1; j < n; j++)
            if (list[j] < list[min])
                min = j;
        SWAP(list[i], list[min], temp);
    }
}
```



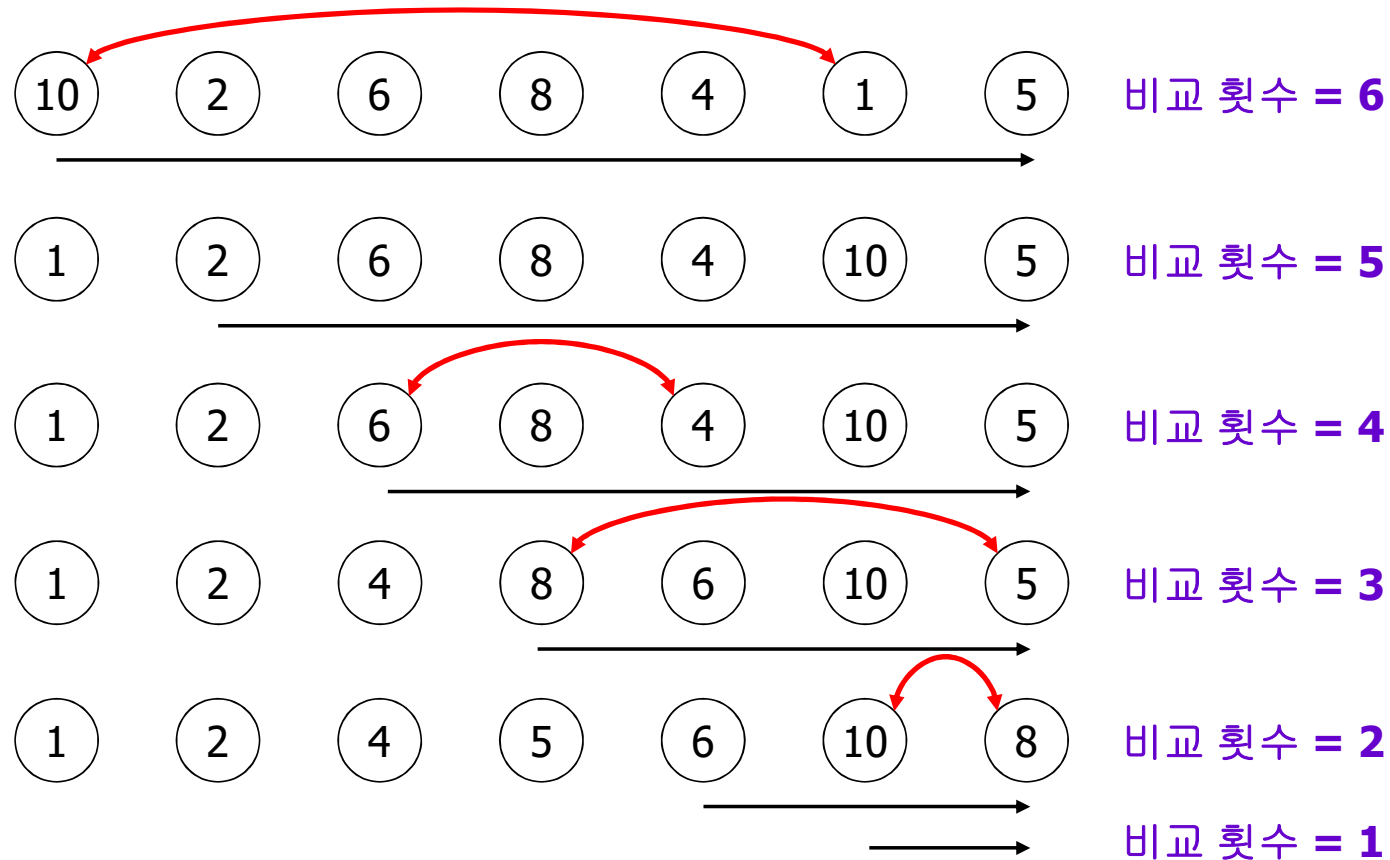
```
// list[i]부터 list[n-1]까지 정렬.
// 최소값이 i에 있다고 일단 가정
// i 위치 다음의 모든 놈들에 대해
// 더 작은 것이 있으면
// 최소값을 이 놈으로...
// 최소값과 i의 내용을 교체
```

Selection Sorting – 정의

- 가장 작은 것부터 찾은 후, 차례대로 저장하자.



Selection Sorting – 분석 (비교 횟수)

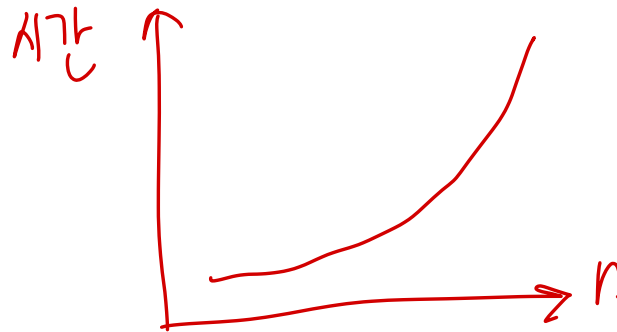


Selection Sorting의 성능

- 데이터 수가 n 개일 때, 비교 연산의 수

$$\sum_{i=1}^{n-1} k = \frac{n(n-1)}{2}$$

n^2 복잡도



예 2: 이진 검색(Binary Search)

정렬하는 이유: 빨리 보기 위해

■ 문제

- 가정: 서로 다른 $n \geq 1$ 개의 정수가 `list[]` 배열에 정렬
- `key`가 주어질 때, `list[i] = key`인 `i`를 발견하여 출력.

$\log_2 n$

$n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \dots |$

■ 해결 방법

■ Step 1

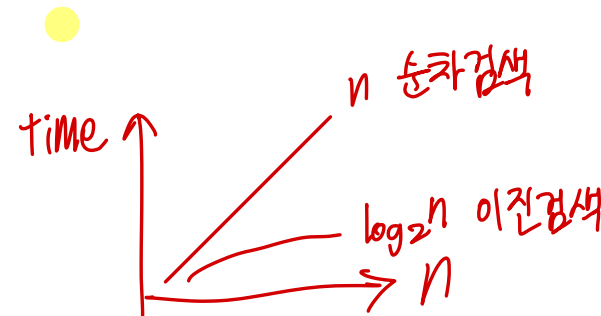
- `left = 0. right = n-1. middle = (left+right)/2`로 설정

■ Step 2

- `list[middle]`과 `key`를 비교

■ 초기 알고리즘: Program 1.5

■ 최종 알고리즘: Program 1.7



Program 1.5: 이진 검색의 초기 해

```
while ( there are more integers to check ) {  
    middle = ( left + right ) / 2;  
    if ( key < list [ middle ] )  
        right = middle - 1;  
    else if ( key == list [middle] )  
        return middle;  
    else left = middle + 1;  
}
```

left=0
right=n-1

Program 1.7: 이진 검색의 최종 해

```

int binsearch(int list[], int key, int left, int right)
{
    /*search list[0] <= list[1] <= ... <= list[n-1] for key.
       Return its position if found. Otherwise return -1 */
    int middle;          // left = 0, right = n-1 로 전달
    while (left <= right) {
        middle = (left + right)/2;
        switch (compare(list[middle], key)) {
            case -1: left = middle + 1;    // key가 크다
                     break;
            case 0: return middle;          // 같다
            case 1: right = middle - 1;     // key가 작다
        }
    }
    return -1;
}
    
```

Program 1.6: Compare의 구현 방법

- **Macro**

```
#define compare(x, y) (((x)<(y)) ? -1 : ((x)==(y)) ? 0 : 1)
```

- 함수 호출

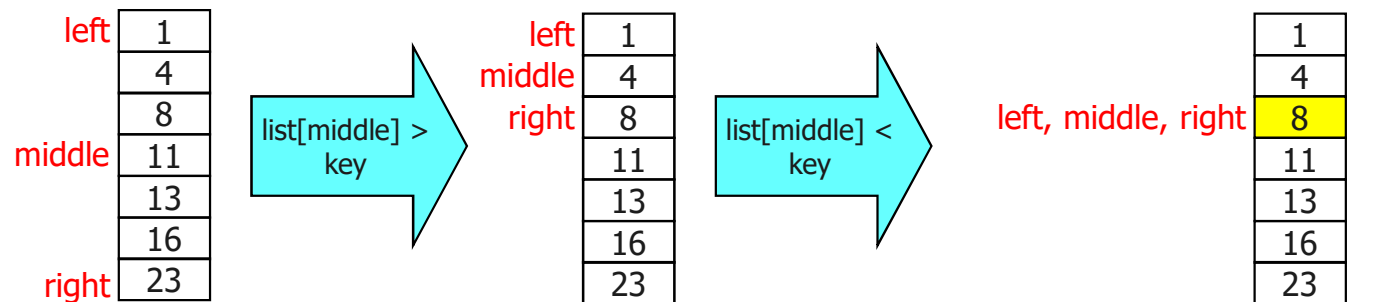
```
int compare( int x, int y )
{
    // x와 y를 비교하여 경우에 따라 -1, 0, 1을 출력
    if ( x < y ) return -1;
    else if ( x == y ) return 0;
    else return 1;
}
```

순차 접근 : 연결 리스트 ← 연산 X

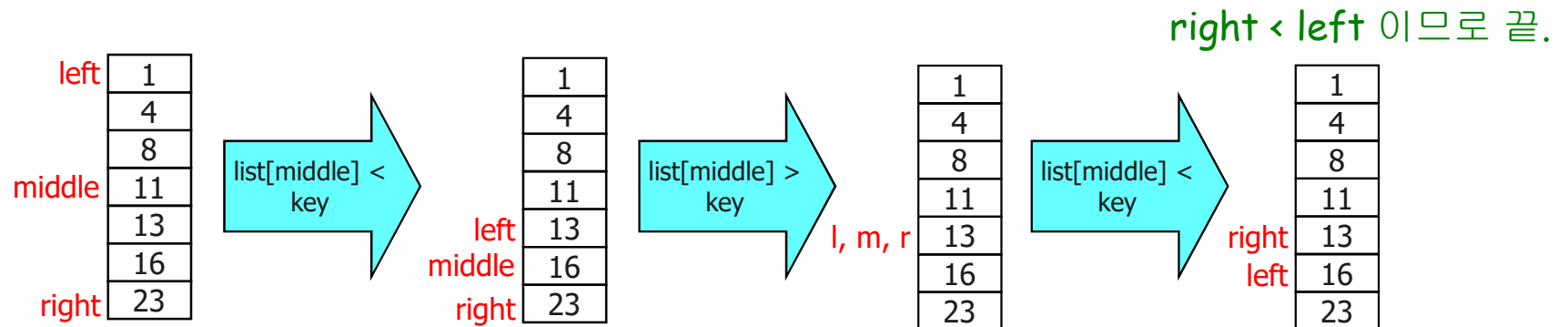
임의 접근 : 이진 탐색

Example: 이진 검색

(1) case 1: key = 8



(2) case 2: key = 15



재귀 알고리즘(Recursive Algorithms)

- 재귀 알고리즘의 특징

- 문제 자체가 재귀적일 경우 적합(예: 피보나치 수열) $F(n) = F(n-1) + F(n-2)$
- 이해하기가 용이하나, 비효율적일 수 있음

- 재귀 알고리즘을 작성하는 방법

- ※ (
 - 재귀 호출을 종료하는 경계 조건을 설정
 - 각 단계마다 경계 조건에 접근하도록 알고리즘의 재귀 호출

- 재귀 알고리즘의 두 가지 예

- 이진 검색
- 순열(Permutations)

이진 검색의 재귀 알고리즘

```
int binsearch(int list[], int key, int left, int right)
{
    /*search list[0] <= list[1] <= ... <= list[n-1] for key.
    Return its position if found. Otherwise return -1 */
    int middle;
    if (left <= right) {
        middle = (left + right) / 2;
        switch (compare(list[middle], key)) {
            case -1: return binsearch(list, key, middle+1, right);
            case 0: return middle;
            case 1: return binsearch(list, key, left, middle-1);
        }
    }
    return -1;
}
```



예 1.4: 순열(Permutations)

- 문제 정의
 - $n \geq 1$ 개의 원소를 갖는 집합에 대해 이 집합의 모든 원소들에 대한 순열을 출력하라.
 - 예: 집합 $\{a, b, c\}$ 에 대해 순열의 집합 =
 - $\{(a, b, c), (a, c, b), (b, a, c), (b, c, a), (c, a, b), (c, b, a)\}$
 - n 개의 원소에 대한 순열의 수는 $n!$
- 재귀적인 해결 방법: 집합 $\{a, b, c, d\}$ 를 가정
 - a 가 먼저 나온 후, $\{b, c, d\}$ 로 구성된 모든 순열
 - b 가 먼저 나온 후, $\{a, c, d\}$ 로 구성된 모든 순열
 - c 가 먼저 나온 후, $\{a, b, d\}$ 로 구성된 모든 순열
 - d 가 먼저 나온 후, $\{a, b, c\}$ 로 구성된 모든 순열

Program 1.9: 순열을 출력하는 재귀함수

```
void perm(char *list, int i, int n)
// list[i]에서 list[n]까지의 원소로 구성된 모든 순열 출력
// {a, b, c, d}의 경우 초기 호출 = perm(list, 0, 3)
{
    int j, temp;
    if (i == n) { // 단 하나의 순열만 존재. 그냥 출력하자...
        for (j = 0; j <= n; j++)
            printf("%c", list[j]);
        printf("\n");
    }
    else { // 하나 이상의 순열 존재. 재귀적으로 출력
        for (j = i; j <= n; j++) {
            SWAP(list[i], list[j], temp);
            perm(list, i+1, n);
            SWAP(list[i], list[j], temp);
        }
    }
}
```

$$\begin{aligned} T(0, n) &= (n+1) \cdot T(1, n) \\ T(1, n) &= n \cdot T(2, n) \\ T(n, n) &= 1 \cdot (n+1) \\ &\downarrow \\ (n+1)! \cdot (n+1) \\ &O(n^2 \times n!) ? \end{aligned}$$

perm() 함수의 동작 과정



$\{a, b, c, d\}$ 초기화 $\text{perm}(\text{list}, 0, 3)$

bcd a 다음에 출력되는 순열

ba cd
bc ad
bcd a
 \therefore bdca



3. 데이터 추상화(Data Abstraction)

- **C** 언어에서 데이터 타입
 - 기본형: char, int, float, double
 - 확장형: short, long, unsigned
 - 그룹화: array, struct, union
 - 포인터
- 데이터 타입의 정의
 - 데이터 객체의 모음 및 그 데이터 객체에 적용 가능한 연산들의 집합
- 예: **int**
 - 객체: {0, 1, -1, 2, -2, ... , INT_MAX, INT_MIN}
 - 연산: {+, -, *, /, %, ...}

추상적 데이터 타입(Abstract Data Type)

- 데이터 객체의 내부 표현양식을 아는 것이 도움이 될까?
 - Yes, but dangerous!
- **Abstract Data Type (ADT)의 정의**
 - 데이터 객체 및 연산의 명세와 데이터 객체의 내부 표현양식/연산의 구현 내용을 분리
 - 예: Ada package, C++ class
- **ADT에서 연산의 명세**
 - 구성 요소: 함수 이름, 인자들의 타입, 결과들의 타입
 - 함수의 호출 방법 및 결과물이 무엇인지를 설명
 - 함수의 내부 동작과정 및 구현 방법은 은폐
- **Information Hiding**



연산 명세에서 내부 함수들의 종류

- **생성자(Creator/Constructor)**
 - 데이터 객체의 새로운 인스턴스 생성
- **Transformer**
 - 기존 인스턴스를 이용하여 새로운 인스턴스를 생성
- **관찰자(Observer/Reporter)**
 - 인스턴스에 대한 정보를 출력

ADT의 예: Natural Number

ADT Natural_Number

객체: 0부터 시작하여 컴퓨터로 표현할 수 있는 최대 정수(INT_MAX)까지의 범위에 속하는 정수들의 집합

함수:

for all $x, y \in \text{Natural_Number}$; $\text{TRUE}, \text{FALSE} \in \text{Boolean}$
and where $+$, $-$, $<$, and $==$ are the usual integer operations

Nat_No	Zero()	::=	0
자연수, 양의 정수	Boolean Is_Zero(x)	::=	if (x) return FALSE else return TRUE
Nat_No	Add(x, y)	::=	if ((x + y) <= INT_MAX) return x + y else return INT_MAX
Boolean	Equal(x, y)	::=	if (x == y) return TRUE else return FALSE
Nat_No	Successor(x)	::=	if (x == INT_MAX) return x else return x + 1
Nat_No	Subtract(x, y)	::=	if (x < y) return 0, else return x - y
end Natural_Number			

4. 성능 분석(Performance Analysis)

■ 프로그램의 평가 기준

- 주어진 문제를 해결
- 정확성(Correctness)
- 문서화(Documentation)
- 모듈화(Modularization)
- 가독성(Readability)
- 공간 효율성(Space efficiency)
- 시간 효율성(Time efficiency)

} 필수적인 요소

} 좋은 프로그래밍 습관

} 성능과 관련

★ 성능 분석(Complexity theory, Simulation) vs.
성능 측정(Benchmarking) 컴퓨터 성능마다 다름

■ 복잡도(Complexity)의 정의

- 공간 복잡도: 프로그램 실행에 소요되는 메모리
- 시간 복잡도: 프로그램의 실행 시간

4.1 공간 복잡도(Space Complexity)

- 고정적인 공간 요구사항
 - 입력과 출력 크기에 무관한 공간들
 - 예: 명령어 공간, 단순 변수나 상수를 위한 공간, ...
- 가변적인 공간 요구사항: $S_p(I)$
 - I 의 수나 크기, 그리고 I/O 의 횟수 등에 따라 가변적인 공간
- 프로그램 P 의 전체 공간 요구 $S(P) = c + S_p(I)$
- 예: 단순 산술 함수 $S_{abc}(I) = 0$ 고정에 따라

```
float abc( float a, float b, float c )    {  
    return a+b+b*c + (a+b-c)/(a+b) + 4.0;  
}
```
- 예: 배열에 저장된 원소들의 합: **Program 1.11**

Program 1.11과 Program 1.12

✗ 커도 작아도 상관X
고정 메모리 0
주소, 정수 하나씩

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

Program 1.11: $S_{\text{sum}}(I) = 0$

가변적 메모리

```
float rsum(float list[], int n)
{
    if (n) return rsum(list, n-1) + list[n-1];
    return 0;
}
```

주소 정수
↓ ↓

Program 1.12: $S_{\text{rsum}}(I) = (n+1)(\text{float} * + \text{int})$

함수호출 횟수


4.2 시간 복잡도(Time Complexity)

- $T_p = \text{컴파일 시간} + \text{실행 시간}$ 더 중요
 - 컴파일 시간은 고정 & 한번만 필요
 - 질문: T_p 를 어떻게 계산할까?
 - T_p 는 컴파일러 option과 하드웨어 사양에 따라 가변
 - 프로그램 단계 수(Program Step)을 활용하자.
- **Program Step**
 - 정의: 실행 시간이 프로그램의 특성과는 무관한 프로그램의 문법적인 혹은 논리적인 단위
 - Program step의 계산: count를 이용
 - 예: Program 1.13 ~ Program 1.18



Program 1.13: Count 이용 예(1)

```
float sum(float list[], int n)
{
    float tempsum = 0; count++; // for assignment
    int i;
    for (i = 0; i < n; i++) {
        count++; // for the for loop
        count++; // for assignment
        tempsum += list[i];
    }
    count++; // last execution of for
    count++; // for return
    return tempsum;
}
```



Program 1.14: 1.13의 단순화

```
float sum(float list[], int n)
{
    float tempsum = 0;
    int i;
    for (i = 0; i < n; i++)
        count += 2;
    count += 3;
    return 0;
}
```

Program 1.15: Count 이용 예(2)

```
float rsum(float list[], int n)
{
    count++; // for if conditional
    if (n) {
        count++; // for return and rsum invocation
        return rsum(list, n-1) + list[n-1];
    }
    count++;
    return list[0];
}
```



Program 1.16: 행렬 더하기

```
void add(int a[][MAX_SIZE], int b[][MAX_SIZE],
         int c[][MAX_SIZE], int rows, int cols)
{
    int i, j;
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            c[i][j] = a[i][j] + b[i][j];
}
```

Program 1.17: Count 이용 예(3)

```
void add(int a[][MAX_SIZE], int b[][MAX_SIZE],
         int c[][MAX_SIZE], int rows, int cols)
{
    int i, j;
    for (i = 0; i < rows; i++) {
        count++;           // for i for loop
        for (j = 0; j < cols; j++) {
            count++;       // for j for loop
            c[i][j] = a[i][j] + b[i][j];
            count++;       // for assignment statement
        }
        count++;           // last time of j for loop
    }
    count++;               // last time of i for loop
}
```



Program 1.18: 1.17의 단순화

```
void add(int a[][MAX_SIZE], int b[][MAX_SIZE],
         int c[][MAX_SIZE], int rows, int cols)
{
    int i, j;
    for (i = 0; i < rows; i++)
        for (j = 0; j < cols; j++)
            count += 2;
    count += 2;
}
count++;
}
```




Step Count Table (1)

Statement	s/e	Frequency	Total step
Float sum(float list[], int n)	0	0	0
{	0	0	0
float tempsum = 0;	1	1	1
int i;	0	0	0
for (i = 0; i < n; i++)	1	n+1	n+1
tempsum += list[i];	1	n	n
return tempsum;	1	1	1
}	0	0	0
Total			2n+3



Step Count Table (2)

Statement	s/e	Frequency	Total step
Float rsum(float list[], int n)	0	0	0
{	0	0	0
if (n)	1	n+1	n+1
return rsum(list, n-1)+ list[n-1];	1	n	n
return list[0];	1	1	1
}	0	0	0
Total			2n+2

Step Count Table (3)

Statement	s/e	Frequency	Total step
void add(int a [] [MAX_SIZE] ...)	0	0	0
{	0	0	0
int i, j;	0	0	0
for (i = 0; i < rows; i++)	1	rows+1	rows+1
for (j = 0; j < cols; j++)	1	rows · (cols+1)	rows · cols + rows
c[i][j] = a[i][j] + b[i][j];	1	rows · cols	rows · cols
}	0	0	0
Total			2rows · cols + 2rows + 1

step: n의 다항식 형태

4.3 근사 표현(O , Ω , Θ)

1장에서 제일 중요

■ 동기

- 정확한 step count를 계산하는 것은 쉽지 않다.
- Program step의 정의 자체가 정확하지 않다.
- $100n + 10$ 과 $30n + 30$ 의 비교

■ 접근 방법

- $T_p(n) = c_1n^2 + c_2n$ 이라고 가정
- n 이 충분히 클 경우, 임의의 c_3 에 대해 $T_p(n) > c_3n$

■ 정의: $f(n) = O(g(n))$ iff

$$100n + 10 = O(n)$$

- $\exists(c \text{ and } n_0 > 0) \text{ such that}$
- $f(n) \leq cg(n) \text{ for all } n, n \geq n_0.$

$$n^3 + 10^6 n^2 = O(n^3)$$

$$n^3 + 10^6 n^2 \leq c \cdot n^3, \quad n \geq n_0 \quad c? \quad c=10 \quad n=10^6$$

$$\frac{10^6 + 1}{10 \cdot 1} \cdot n^3 \quad c=10 \quad \underline{O(n^3)} //$$

$$10^{17} + 10^6 \cdot 10^{12} \leq 10 \cdot 10^6 \quad n=10^6$$

$$10 \cdot 10^{17} = 10^{19}$$

근사 표현(1): $f(n) = O(g(n))$

■ Example

- $3n + 2 = O(n)$ as $3n + 2 \leq 4n$ for all $n \geq 2$
- $3n + 3 = O(n)$ as $3n + 3 \leq 4n$ for all $n \geq 3$
- $100n + 6 = O(n)$ as $100n + 6 \leq 101n$ for all $n \geq 6$
- $10n^2 + 4n = O(n^2)$ as $10n^2 + 4n \leq 11n^2$ for all $n \geq 5$
- $6 \cdot 2^n + n^2 = O(2^n)$ as $6 \cdot 2^n + n^2 \leq 7 \cdot 2^n$ for all $n \geq 4$
- $3n + 2 \neq O(1)$ & $10n^2 + 4n + 2 \neq O(n)$

↑ $O(1)$
↓

나와 상관 X, 실행 시간 항상 일정, 상수

$100n^2 + 4n = O(n)$ ✗
 $100n^2 + 4n \leq C \cdot n \quad n \geq n_0$
 ↑
 101
 ↗
 $\frac{100n^2}{n} = 100n$

■ $f(n) = O(g(n))$ 일 경우, $g(n)$ 은 $f(n)$ 의 upper bound

- ✱ ■ $10n^2 + 4n = O(n^4)$ as $10n^2 + 4n \leq 10n^4$ for all $n \geq 2$

■ Theorem: If $f(n) = a_m n^m + \dots + a_1 n + a_0$, then $f(n) = O(n^m)$.

- Proof: $\sum_{i=0}^m |a_i| n^i \leq n^m \sum_{i=0}^m |a_i| n^{i-m} \leq n^m \sum_{i=0}^m |a_i|$, for all $n \geq 1$

근사 표현(2): $f(n) = \Omega(g(n))$ *Omega*

- 정의: $f(n) = \Omega(g(n))$ iff
 - $\exists(c \text{ and } n_0 > 0)$ such that $f(n) \geq cg(n)$ for all $n, n \geq n_0$.
 - $g(n)$ 은 $f(n)$ 의 lower bound
- **Example**
 - $3n + 2 = \Omega(n)$ as $3n + 2 \geq 3n$ for all $n \geq 1$
 - $10n^2 + 4n = \Omega(n^2)$ as $10n^2 + 4n \geq 10n^2, \forall n \geq 1$
 - $6 \cdot 2^n + n^2 = \Omega(2^n)$ as $6 \cdot 2^n + n^2 \geq 6 \cdot 2^n, \forall n \geq 1$
 - $3n + 2 = \Omega(1), 10n^2 + 4n = \Omega(n), 6 \cdot 2^n + n^2 = \Omega(n^2)$
- **Theorem:** If $f(n) = a_m n^m + \dots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.

근사 표현(3): $f(n) = \Theta(g(n))$

Theta

- 정의: $f(n) = \Theta(g(n))$ iff
 - $\exists (c_1, c_2, \text{ and } n_0 > 0)$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n, n \geq n_0$.
 - $g(n)$ 은 $f(n)$ 의 lower bound이면서 upper bound임
- **Example**
 - $3n + 2 = \Theta(n)$ as $3n + 2 \geq \overset{c_1}{3}n$ for all $n \geq \overset{n_0}{2}$ and $3n + 2 \leq \overset{c_2}{4}n$ for all $n \geq 2$
 - $10n^2 + 4n + 2 = \Theta(n^2)$, $6 \cdot 2^n + n^2 = \Theta(2^n)$
 - $3n + 2 \neq \Theta(1)$, $3n + 2 \neq \Theta(n^2)$
- **Theorem:** If $f(n) = a_m n^m + \dots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.

예제 프로그램들의 근사 표현

- $T_{\text{sum}}(n) = 2n + 3 = \Theta(n)$
- $T_{\text{rsum}}(n) = 2n + 2 = \Theta(n)$
- $T_{\text{add}}(\text{row}, \text{col}) = 2\text{row} * \text{col} + 2\text{row} + 1 = \Theta(\text{row} * \text{col})$ $\leftarrow \Theta(n^2)$

$$A(n)(n) + B(n)(n) = (n)(n)$$

- Binary Search: $\Theta(\log_2 n)$ \leftarrow see Program 1.6

- $T(n) = T(n/2) + 1$

- $T(1) = 1$ 점화식

$$T(n) = T\left(\frac{n}{2}\right) + 1 \rightarrow \Theta(\log_2 n)$$

$$T\left(\frac{n}{4}\right) + 1$$

$$T\left(\frac{n}{8}\right) + 1$$

$$\vdots$$

$$T(1) + 1$$

- Permutation: $\Theta(n^2 * n!)$ \leftarrow see Program 1.8

- $T_{\text{perm}}(0, n) = (n + 1) * T_{\text{perm}}(1, n)$

- $T_{\text{perm}}(n, n) = n + 1$

- Magic Square (Figure 1.6 & Program 1.23)

예: Magic Square

- **Magic Square**의 정의

- $n \times n$ 행렬에 1부터 n^2 까지의 정수를 채우는데, 각 행과 열, 그리고 대각선의 합이 모두 동일.

- **$n = 5$ 인 magic square (합은 65)**

input

output

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

6	1	8
7	5	3
2	9	4

Program 1.23: Magic Square (1)

```
#include <stdio.h>
#define MAX_SIZE 15           // Square의 최대 크기
void main(void)               // 순차적으로 magic square 작성
{
    int square[MAX_SIZE][MAX_SIZE];
    int i, j, row, column;    // 배열의 첨자들
    int size;                  // Square의 크기로 입력 받음.
    int count;                 // 1부터 size * size까지 증가

    printf("Enter the size of the square: ");
    scanf("%d", &size);
    if (size < 1 || size > MAX_SIZE + 1) { // 입력 오류 검사
        printf("Error! Size is out of range\n");
        return;
    }
}
```

영향 X

Program 1.23: Magic Square (2)

명함 X { if (!(size % 2)) { // square의 크기는 홀수여야 함.
print("Error! Size is even\n");
return;
}

명함 0 { for (i = 0; i < size; i++) // square의 모든 원소를
for (j = 0; j < size; j++) // 0으로 초기화
square[i][j] = 0;
size² square[0][size / 2] = 1; // 첫 행의 중간부터 시작
i = 0; // i는 현재 행 번호
j = size / 2; // j는 현재 열 번호

size² { for (count = 2; count <= size * size; count++) {
row = (i-1 < 0) ? (size-1) : (i-1); // 위쪽 행
column = (j-1 < 0) ? (size-1) : (j-1); //왼쪽 열
if (square[row][column] != 0) {
i = (++i) % size; // 못 갈 경우, 아래로
}
}

Program 1.23: Magic Square (3)

```
else {           // 갈 수 있을 경우, i와 j를 대각선 위로.
    i = row;
    j = column;
}
square[i][j] = count; // 변경된 위치에 다음 수 추가
}
// 생성된 magic square를 출력
printf("Magic Square of size %d : \n\n", size);
for (i = 0; i < size; i++) {
    for (j = 0; j < size; j++) {
        printf("%5d", square[i][j]);
        printf("\n");           // 한 줄에 한 행씩 출력
    }
    printf("\n\n");
}
```

size²

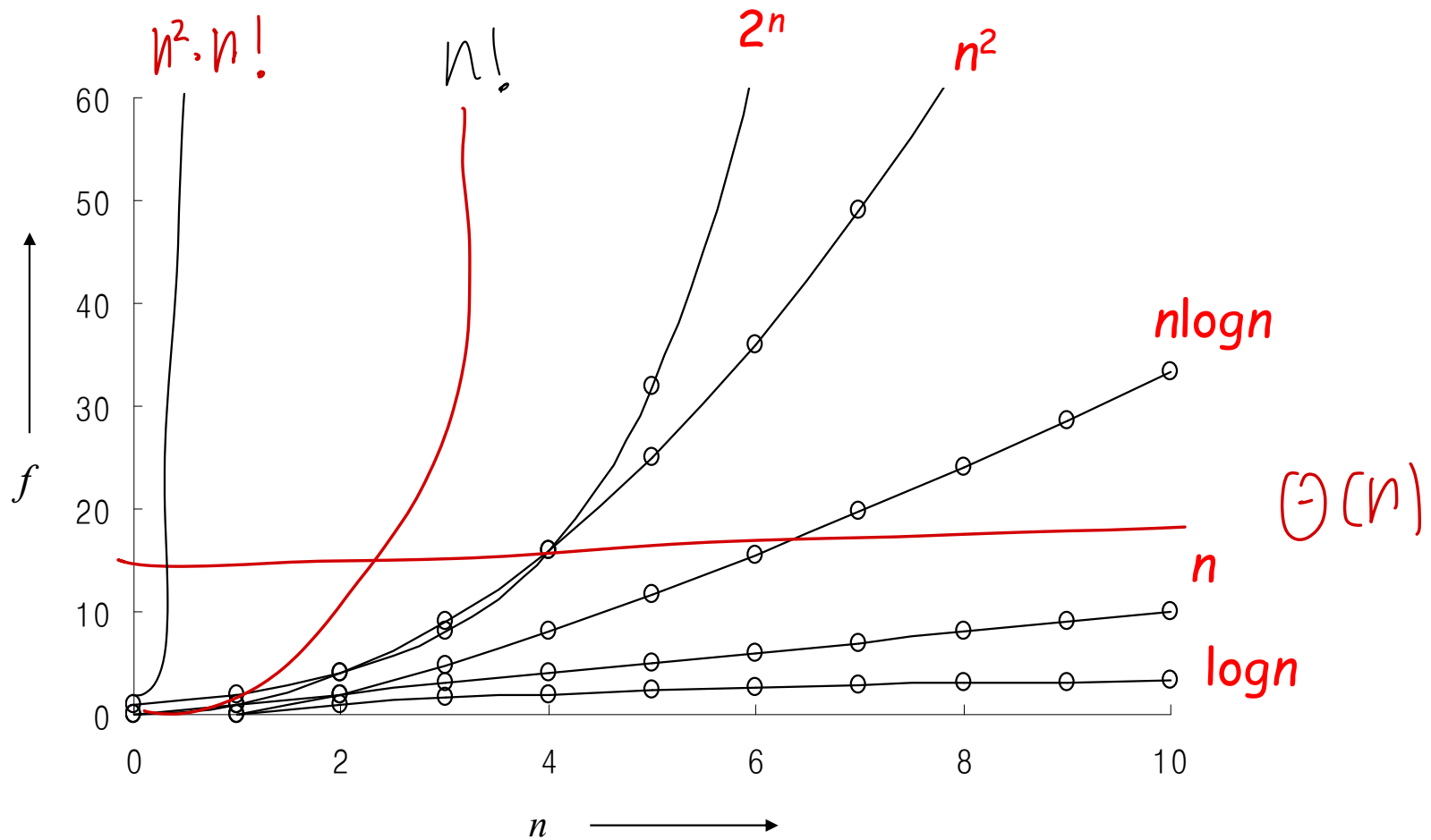
Time Complexity = $\Theta(\text{size}^2)$



5. 성능 측정(Performance Measurement)

- 알고리즘의 복잡도 함수를 그래프로 표시
- **Selection Sort**의 실제 실행 시간을 측정한 후 그래프로 표시

대표적인 복잡도 함수의 그래프




Program 1.24: Selection sort의 시간 측정(1)

```
#include <stdio.h>
#include <time.h>
#define MAX_SIZE 1601
#define ITERATIONS 26
#define SWAP(x, y, t) ((t) = (x), (x) = (y), (y) = (t))
void main(void)      // 다양한 배열 크기에 대해
{                    // Selection sort의 실행 시간을 측정
    int i, j, position;
    int list[MAX_SIZE];
    int sizelist[] = {0, 10, 20, 30, 40, 50, 60, 70, 80,
                      90, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000,
                      1100, 1200, 1300, 1400, 1500, 1600}; // 배열의 크기
    clock_t start, stop;      // 시작 시간과 종료 시간
    double duration;          // 경과 시간 = stop - start
    printf("    n    time\n");
```


Program 1.24: Selection sort의 시간 측정(2)

```
for (i = 0; i < ITERATIONS; i++) { // 26개의 배열 크기
    for (j = 0; j < sizelist[i]; j++)
        list[j] = sizelist[i] - j; // 역순으로 초기에 저장
    ✱ start = clock(); // 시작 시간 측정
    sort(list, sizelist[i]); // sort 함수 호출
    ✱ stop = clock(); // 종료 시간 측정
    // CLOCKS_PER_SEC = 초당 클럭의 수. Macro: 1000
    duration = ((double) (stop - start)) / CLOCKS_PER_SEC;
    printf("%6d %.2f\n", sizelist[i], duration);
}
}
```

Selection Sort의 시간 복잡도는?



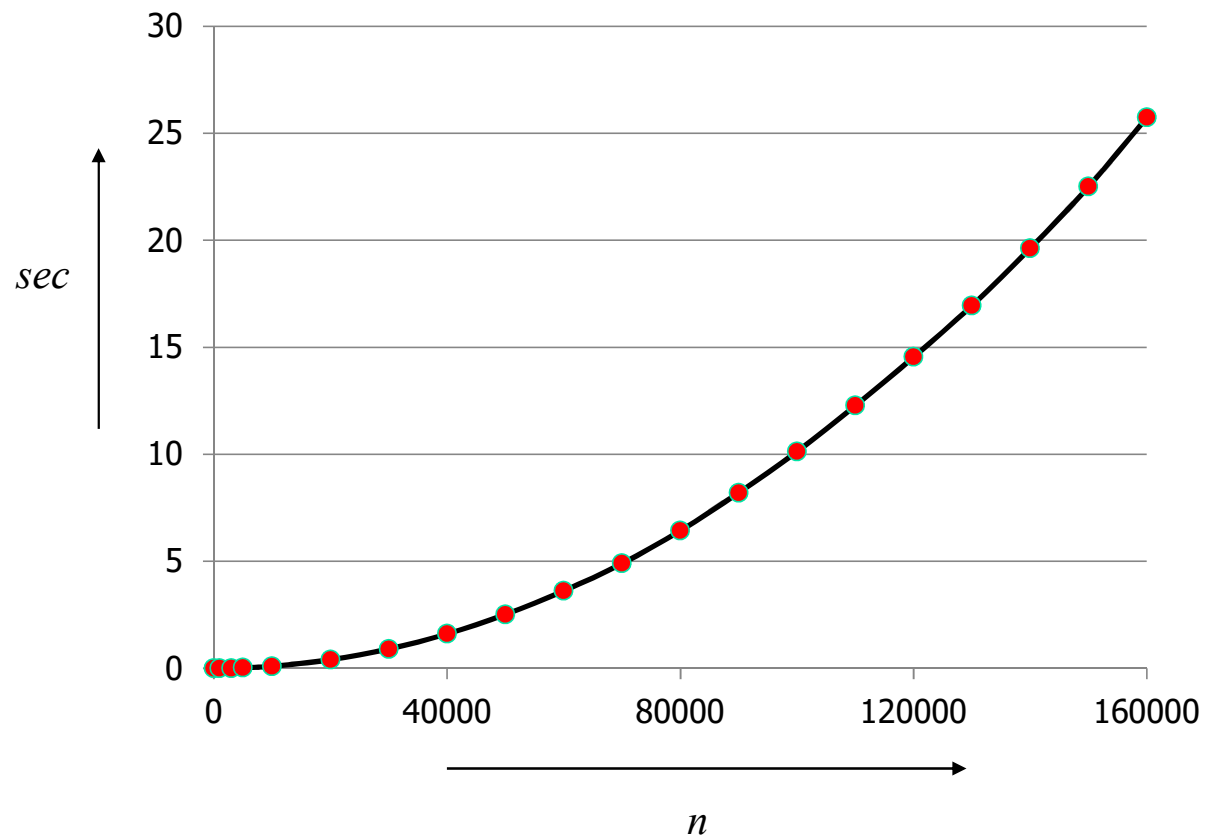
Program 1.24의 실행 결과

n	Time	n	Time
30 ··· 100	.00	900	1.86
200	.11	1000	2.31
300	.22	1100	2.80
400	.38	1200	3.35
500	.60	1300	3.90
600	.82	1400	4.54
700	1.15	1500	5.22
800	1.48	1600	5.93

- **Note:** 요즘 컴퓨터로 실행할 경우...

실행 결과를 그래프로 그리면?

Intel Core i7-4790 @ 3.60GHz, 8.00GB RAM



$\Theta(n^2)$

데이터 크기와
저장된 메모리

시간 복잡도
자료형 선언 X

O(1) 입력 데이터

하노이 타워 시간 복잡도

$$t(n) = 2t(n/4) \quad n > 1 \quad \Theta(n)$$

$$t(1) = 1 \quad \Theta(1)$$

$n = 4^k$, k 는 음이 아닌 수

$$t(n) = 2t\left(\frac{n}{4}\right) = 2 \cdot 2t\left(\frac{n}{16}\right) = 2^2 t\left(\frac{n}{4}\right)$$

$$2^3 + \left(\frac{n}{4^3}\right) \quad \swarrow 4^k$$

$$= 2^k \cdot t\left(\frac{n}{4^k}\right)$$

$$= 2^k$$

$$t(n) = \Theta(\sqrt{n})$$

$$\sum_{i=1}^n \log_2 i$$

$$= \log_2 1 + \log_2 2 + \log_2 3 + \dots + \log_2 n$$

$$= \log_2 (1 \times 2 \times 3 \times \dots \times n)$$

$$= \log_2 n! < \log_2 n^n$$

$$O(n \cdot \log_2 n)$$

점차적 방법

$$O(\log n) \times O(n^2) = O(n^2 \log n)$$

가파른, 완만한, 처음부터 키 큰 불 (등터)