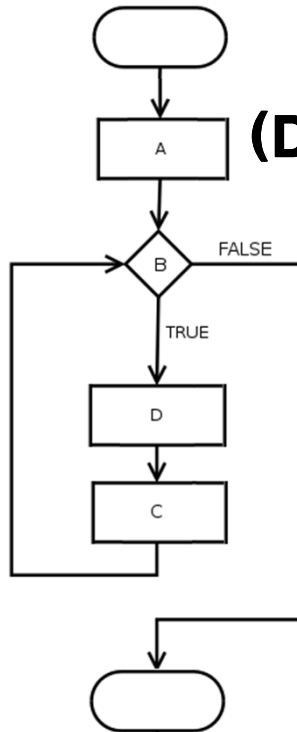


자료구조 & 알고리즘

for(A;B;C)
D;

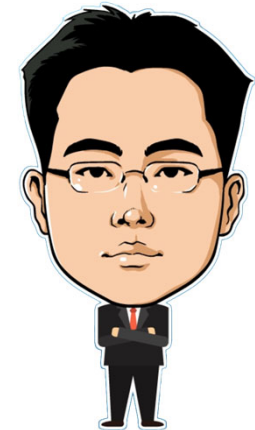


자료구조와 알고리즘 기초 (Data Structures and Algorithms Basics)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



백문이불여일타(百聞而不如一打)



목 차



백문이불여일타(百聞而不如一打)

- 자료구조와 알고리즘
- 재귀와 귀납적 사고
- 정렬과 검색 알고리즘



자료구조와 알고리즘



- 자료구조와 알고리즘

백문이불여일타(百聞而不如一打)

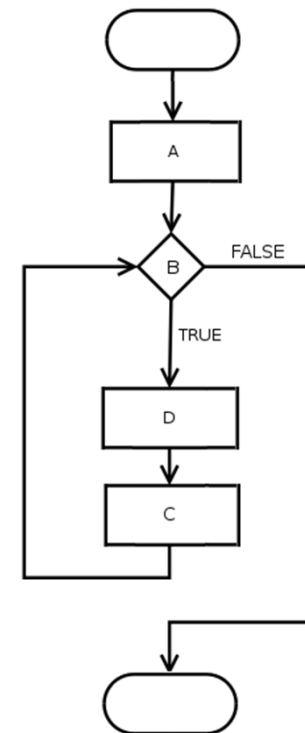
- 알고리즘 복잡도

- 시간 복잡도와 공간 복잡도

- 재귀와 귀납적 사고

- 정렬과 검색 알고리즘

for(A;B;C)
D;



자료구조와 알고리즘 (1/5)

● 자료구조와 알고리즘

○ 자료구조

- 배열(Array), 연결 리스트(Linked List)
- 스택(Stack)과 큐(Queue)
- 트리와 힙(Tree & Heap)
- 해시 테이블(Hash Table)
- 그래프(Graph)

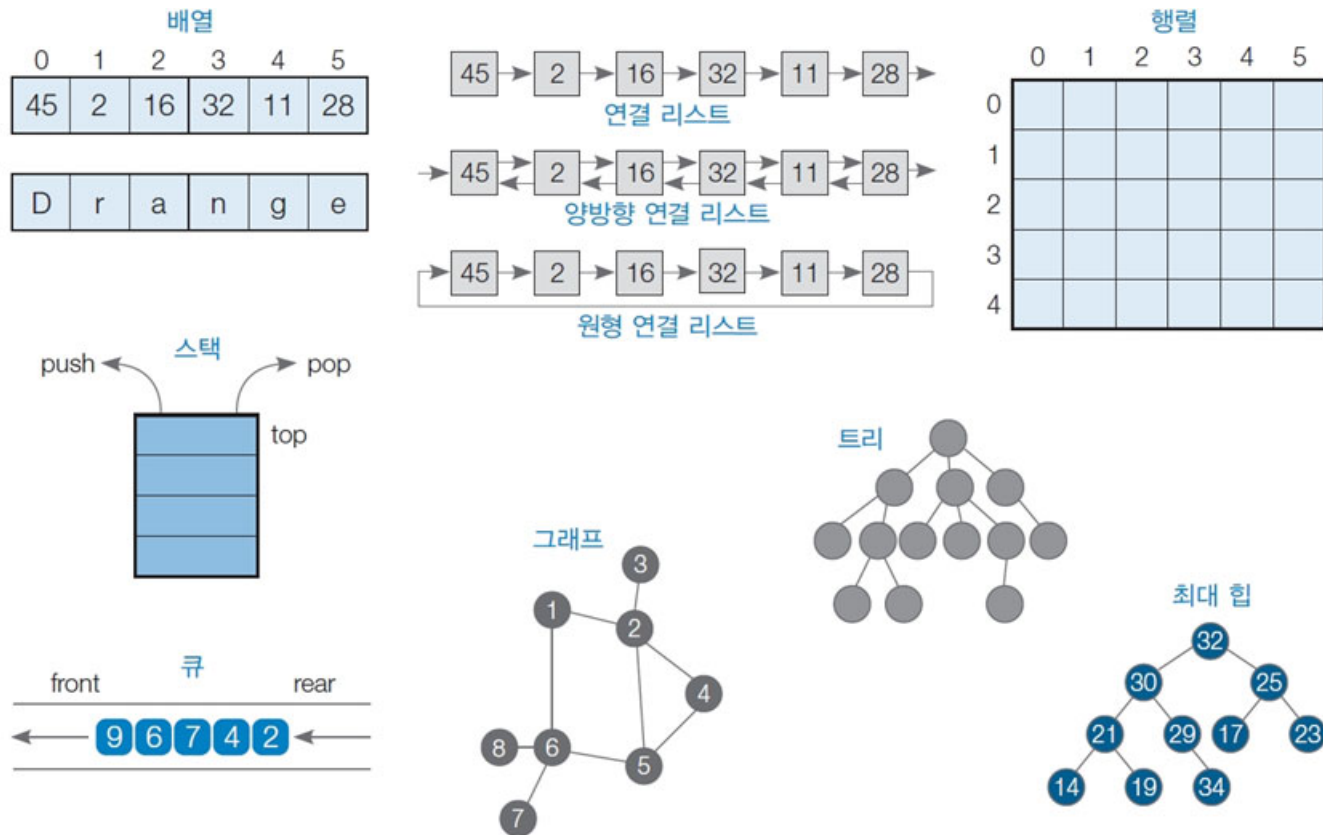
*Algorithms
+ Data Structures
= Programs*

○ 알고리즘

- 재귀(Recursion), 정렬과 검색(Sorting & Search)
- 깊이 우선 탐색(DFS)와 너비 우선 탐색(BFS)
- 탐욕 알고리즘(Greedy Algorithms)
- 동적 프로그래밍(Dynamic Programming)
- 백 트래킹(Backtracking)

자료구조와 알고리즘 (2/5)

● 자료구조의 종류



[이미지 출처: "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.]

자료구조와 알고리즘 (3/5)

● 알고리즘(Algorithm)

“잘 정의된 문제 해결 과정”

- 주어진 문제를 해결하기 위한 잘 정의된 동작들의 유한 집합
 - 어떤 작업을 수행하기 위해 입력을 받아 원하는 출력을 만들어내는 과정을 기술
 - 문제를 풀거나 작업을 수행하기 위한 단계적인 방법
- 자료구조와 알고리즘
 - 자료구조: 자료(행위의 객체: 무엇을)
 - 알고리즘: 문제 해결의 방법(행위적인 측면: 어떻게 하라)

“무엇을 어떻게 하라”



자료구조와 알고리즘 (4/5)

- **알고리즘: 알고리즘의 조건**

- **알고리즘:** 도널드 커누스(Donald Knuth)의 정의
 - **입력을 기반으로 출력을 생성하는 명확하고, 효율적이며 유한한 프로세스**
- **알고리즘의 조건**
 1. **명확함(Definiteness):** 각 단계가 명료하고 간결하며 모호하지 않음.
 2. **효율성(Effectiveness):** 각 동작이 문제 해결에 기여함.
 3. **유한함(Finiteness):** 알고리즘이 유한한 단계를 거친 후 종료됨
 4. **입력(Input):** 외부에서 제공하는 0개 이상의 입력이 존재함.
 5. **출력(Output):** 1개 이상의 출력이 존재함.
 6. **정확성(Correctness):** 알고리즘은 입력이 같으면 항상 같은 결과를 내야 하며, 이 결과가 알고리즘이 해결하는 문제의 정확한 답이어야 한다

**알고리즘은 명확하고
효율적으로 설계해야 한다.**

자료구조와 알고리즘 (5/5)

- **수학적 알고리즘: 1부터 10까지의 합 구하기**

- 1부터 10까지의 합을 구하는 문제를 해결하는 세 가지 알고리즘

1. 1부터 10까지의 숫자를 직접 하나씩 더한다.

$$1 + 2 + 3 + \dots + 10 = 55$$

2. 두수의 합이 10이 되도록 숫자들을 그룹화하여, 그룹의 계수에 10을 곱하고 남은 숫자 5를 더한다.

$$(0 + 10) + (1 + 9) + (2 + 8) + (3 + 7) + (4 + 6) + 5 = 10 \times 5 + 5 = 55$$

3. 공식을 이용하여 계산할 수도 있다.

$$10 \times (1 + 10) / 2 = 55$$

최선의 알고리즘은 어떻게 찾을 것인가?

자료구조와 알고리즘

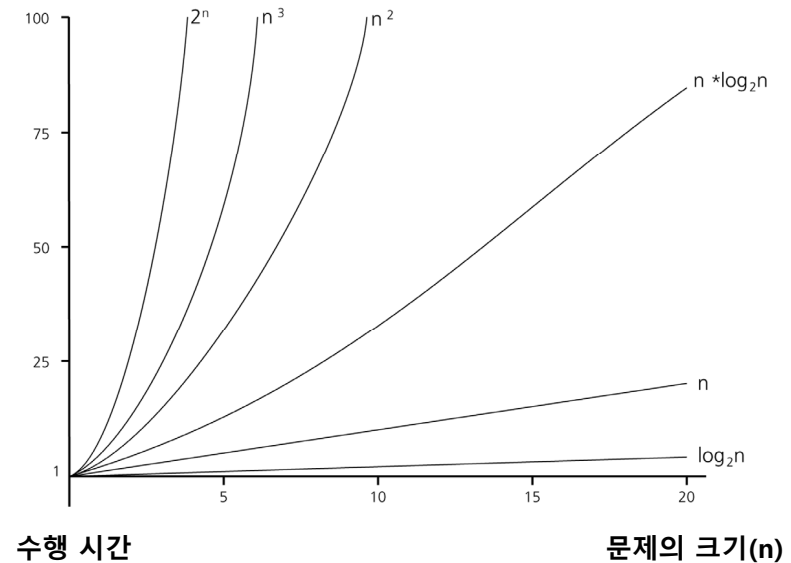
알고리즘 복잡도: 점근적 분석



알고리즘 복잡도: 점근적 분석 (1/5)

● 여러가지 함수의 증가율

- 수행시간은 알고리즘이 수행하는 기본 연산 횟수를 입력 크기에 대한 함수로 표현



| 함수 \ n | 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} | $10^{3,010}$ | $10^{10,103}$ | $10^{301,030}$ |

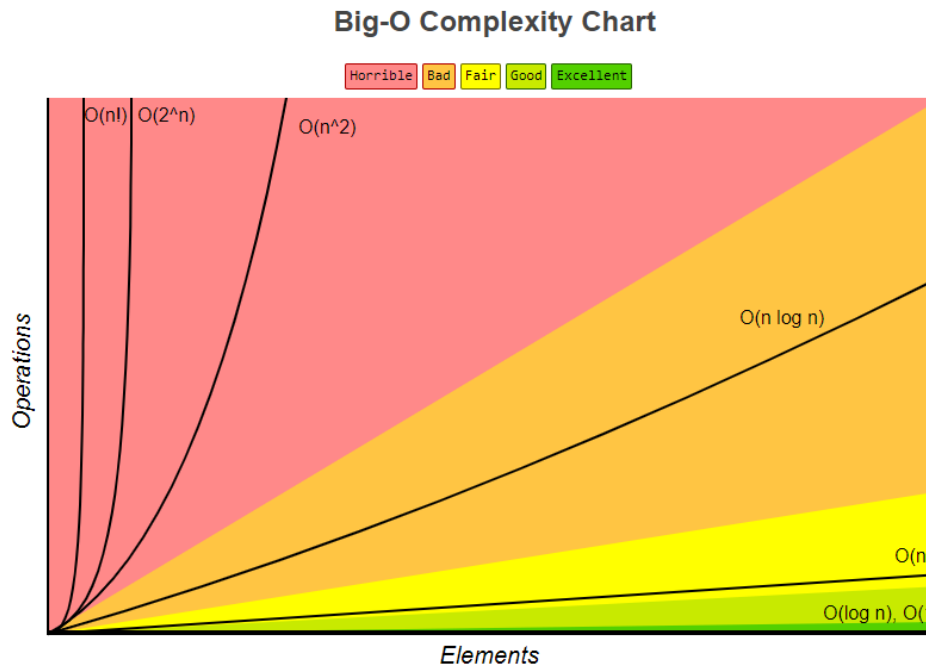
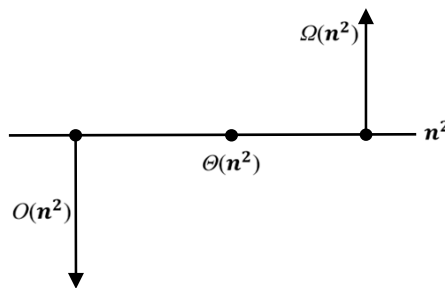
알고리즘 복잡도: 점근적 분석 (2/5)

● 점근적 분석(Asymptotic Analysis)

- 알고리즘의 수행시간을 분석할 때는 항상 입력의 크기가 충분히 큰 때에 대해서 분석한다.

- 점근적 복잡도(Asymptotic Complexity): O , Ω , Θ , o , ω 표기법
- 여러 함수의 증가율과 점근적 개념

$$\lim_{n \rightarrow \infty} f(n)$$



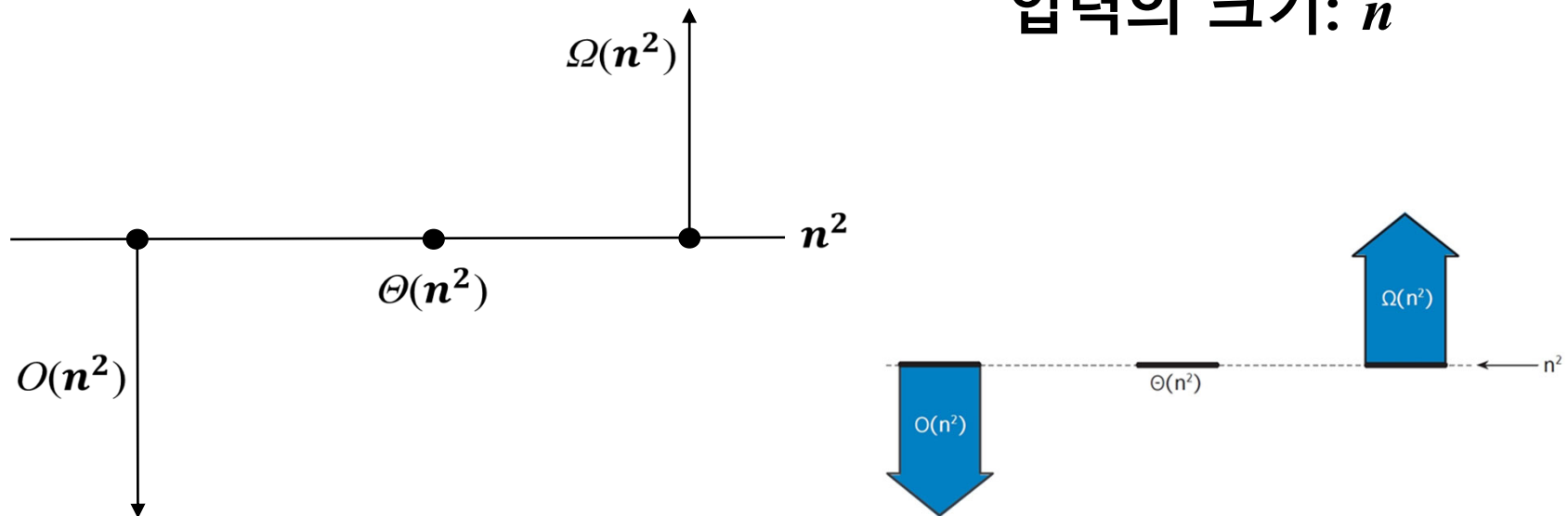
알고리즘 복잡도: 점근적 분석 (3/5)

● 점근적 분석: 점근적 복잡도

○ O , Ω , Θ 표기

- $O(n^2)$: 알고리즘이 **기껏해야(최악)** n^2 에 비례하는 시간(수행 시간의 상한).
- $\Omega(n^2)$: 알고리즘이 **적어도(최선)** n^2 에 비례하는 시간(수행 시간의 하한).
- $\Theta(n^2)$: 알고리즘이 **항상** n^2 에 비례하는 시간이 든다.

입력의 크기: n



알고리즘 복잡도: 점근적 분석 (4/5)

● O : Big Oh 표기법

○ 자주 사용되는 함수의 O-표기와 이름

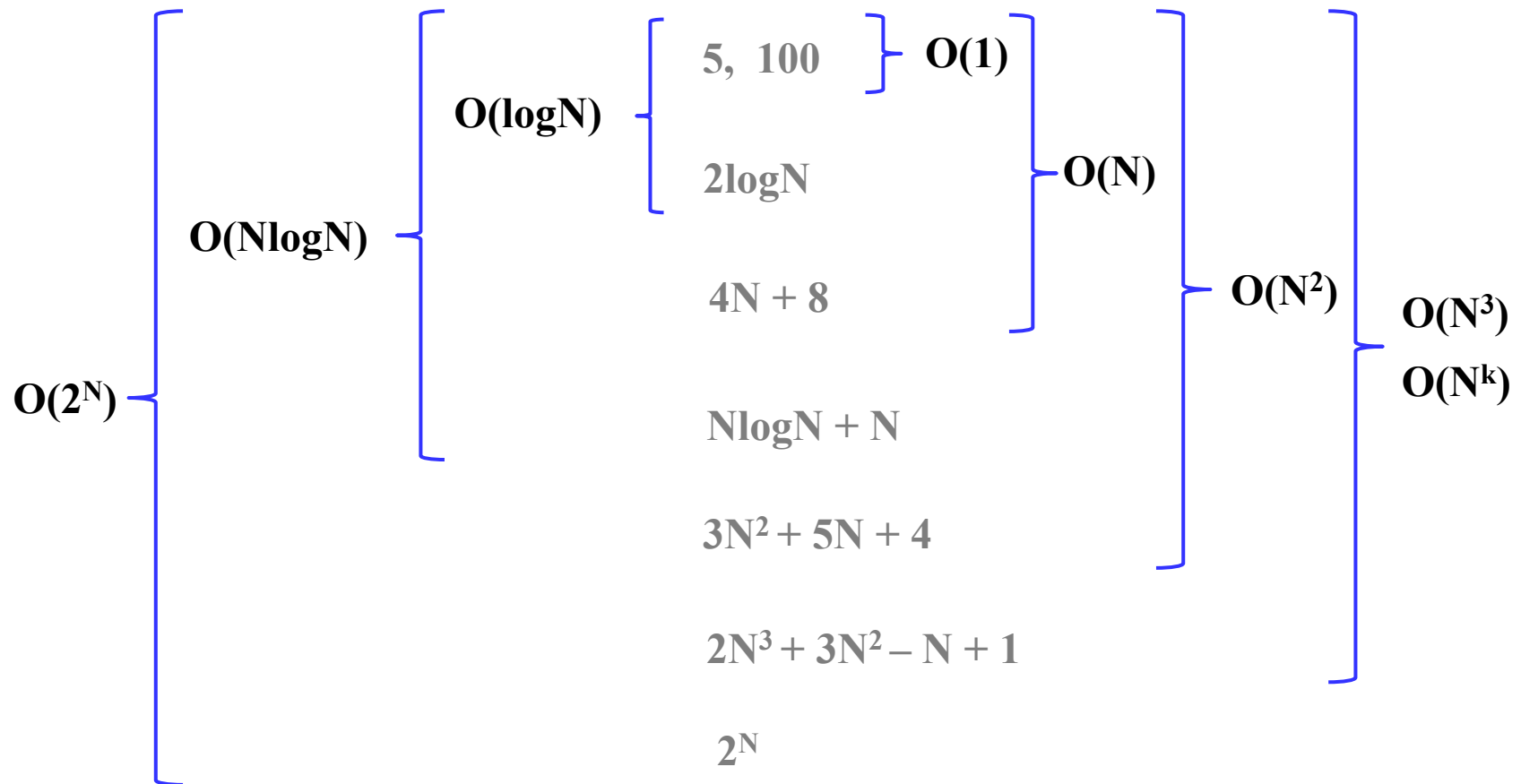
- 알고리즘의 수행시간은 주로 O-표기를 사용하며, 보다 정확히 표현하기 위해 Θ -표기를 사용하기도 한다.

- | | |
|-----------------|---------------------------|
| • $O(1)$ | 상수 시간(Constant Time) |
| • $O(\log N)$ | 로그 시간(Logarithmic Time) |
| • $O(N)$ | 선형 시간(Linear Time) |
| • $O(N \log N)$ | 로그 선형 시간(Log-linear Time) |
| • $O(N^2)$ | 제곱 시간(Quadratic Time) |
| • $O(N^3)$ | 세제곱 시간(Cubic Time) |
| • $O(2^N)$ | 지수 시간(Exponential Time) |

알고리즘 복잡도: 점근적 분석 (5/5)

- O : Big Oh 표기법

- O-표기의 포함 관계



자료구조와 알고리즘

시간 복잡도와 공간 복잡도

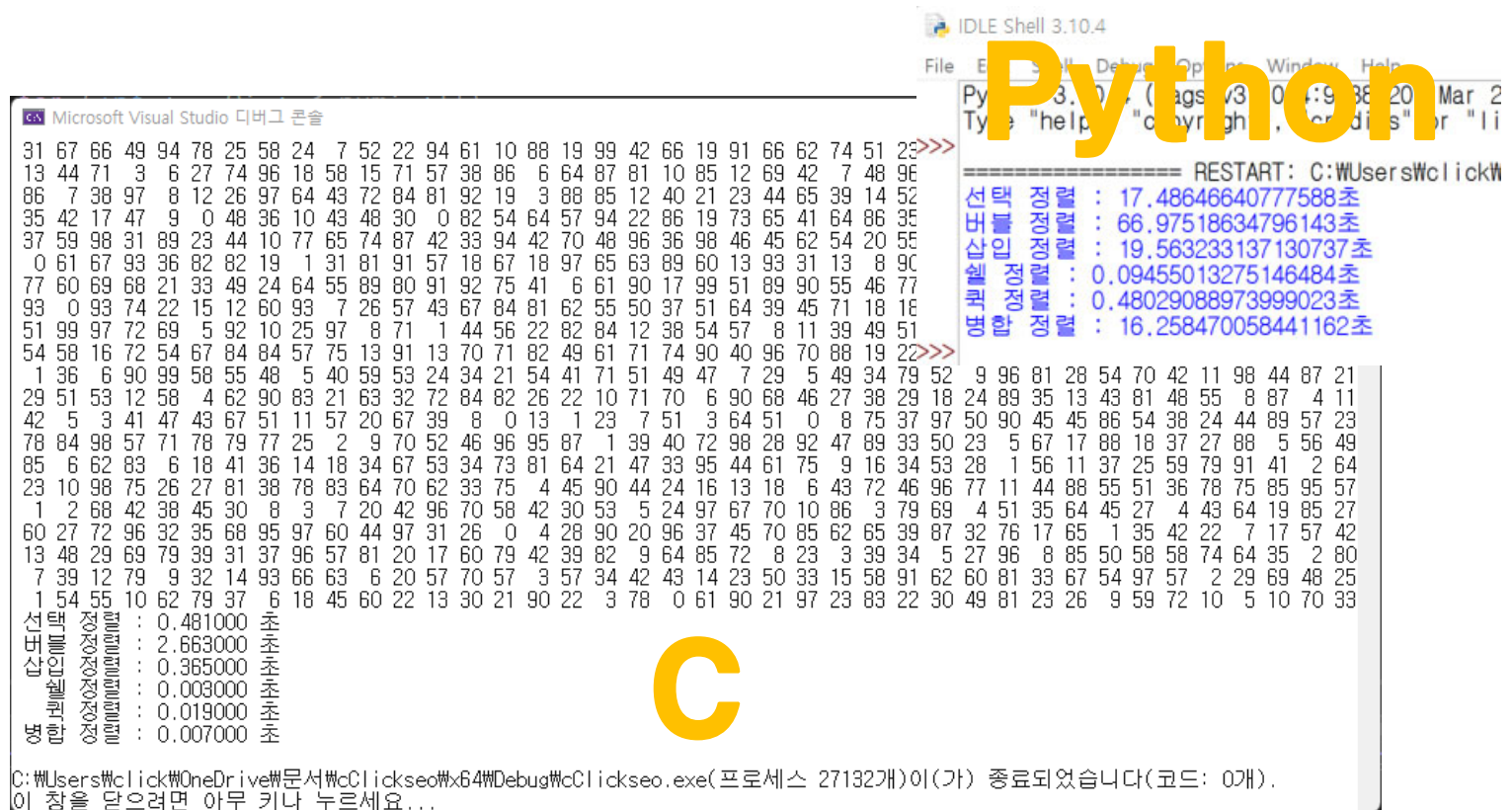


시간 복잡도와 공간 복잡도 (1/3)

● 알고리즘 성능 비교: 경험적 분석

○ 경험적 분석

- 알고리즘을 프로그래밍 언어로 구현 후에 실행 시간을 비교해 보는 것



```
Python 3.10.4 Shell
File Edit Shell Debug Options Window Help
Py 3.10.4 (tags/v3.10.4:9:38:20 Mar 2
Type "help()" for copyright, credits or "li

===== RESTART: C:\Users\WclickW
선택 정렬 : 17.48646640777588초
버블 정렬 : 66.97518634796143초
삽입 정렬 : 19.563233137130737초
셸 정렬 : 0.09455013275146484초
퀵 정렬 : 0.48029088973999023초
병합 정렬 : 16.258470058441162초

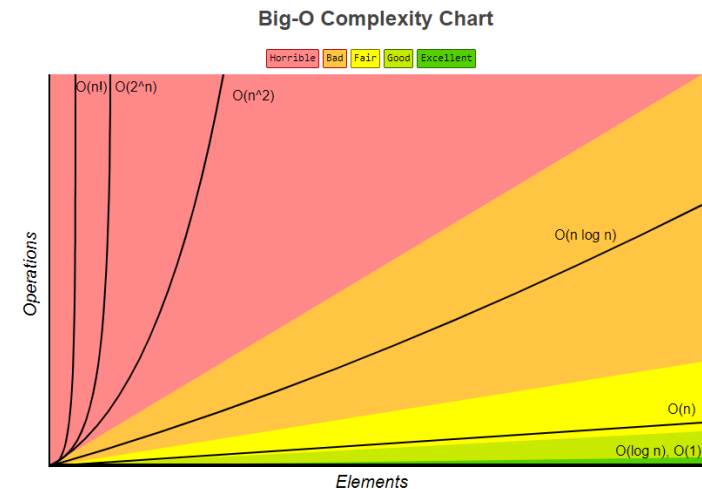
31 67 66 49 94 78 25 58 24 7 52 22 94 61 10 88 19 99 42 66 19 91 66 62 74 51 23
13 44 71 3 6 27 74 96 18 58 15 71 57 38 86 6 64 87 81 10 85 12 69 42 7 48 96
86 7 38 97 8 12 26 97 64 43 72 84 81 92 19 3 88 85 12 40 21 23 44 65 39 14 52
35 42 17 47 9 0 48 36 10 43 48 30 0 82 54 64 57 94 22 86 19 73 65 41 64 86 35
37 59 98 31 89 23 44 10 77 65 74 87 42 33 94 42 70 48 96 36 98 46 45 62 54 20 55
0 61 67 93 36 82 82 19 1 31 81 91 57 18 67 18 97 65 63 89 60 13 93 31 13 8 90
77 60 69 68 21 33 49 24 64 55 89 80 91 92 75 41 6 61 90 17 99 51 89 90 55 46 77
93 0 93 74 22 15 12 60 93 7 26 57 43 67 84 81 62 55 50 37 51 64 39 45 71 18 16
54 58 16 72 54 67 84 84 57 75 13 91 13 70 71 82 49 61 71 74 90 40 96 70 88 19 22
1 36 6 90 99 58 55 48 5 40 59 53 24 34 21 54 41 71 51 49 47 7 29 5 49 34 79 52
29 51 53 12 58 4 62 90 83 21 63 32 72 84 82 26 22 10 71 70 6 90 68 46 27 38 29 18
42 5 3 41 47 43 67 51 11 57 20 67 39 8 0 13 1 23 7 51 3 64 51 0 8 75 37 97
78 84 98 57 71 78 79 77 25 2 9 70 52 46 96 95 87 1 39 40 72 98 28 92 47 89 33 50
85 6 62 83 6 18 41 36 14 18 34 67 53 34 73 81 64 21 47 33 95 44 61 75 9 16 34 53
23 10 98 75 26 27 81 38 78 83 64 70 62 33 75 4 45 90 44 24 16 13 18 6 43 72 46 96
1 2 68 42 38 45 30 8 3 7 20 42 96 70 58 42 30 53 5 24 97 67 70 10 86 3 79 69
60 27 72 96 32 35 68 95 97 60 44 97 31 26 0 4 28 90 20 96 37 45 70 85 62 65 39 87
13 48 29 69 79 39 31 37 96 57 81 20 17 60 79 42 39 82 9 64 85 72 8 23 3 39 34
7 39 12 79 9 32 14 93 66 63 6 20 57 70 57 3 57 34 42 43 14 23 50 33 15 58 91
1 54 55 10 62 79 37 6 18 45 60 22 13 30 21 90 22 3 78 0 61 90 21 97 23 83 22 30
선택 정렬 : 0.481000 초
버블 정렬 : 2.663000 초
삽입 정렬 : 0.365000 초
셸 정렬 : 0.009000 초
퀵 정렬 : 0.019000 초
병합 정렬 : 0.007000 초

C:\Users\WclickW\OneDrive\문서\Wclickseo\64\Debug\Wclickseo.exe (프로세스 27132개)이(가) 종료되었습니다 (코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```


시간 복잡도와 공간 복잡도 (2/3)

● 자료 구조: 시간 복잡도

- 배열 또는 연결 리스트: $O(N)$, 평균 $\Theta(N)$
- 이진 검색 트리: 검색, 삽입, 삭제 시 평균 $\Theta(\log N)$, 최악의 경우 $O(N)$
- 균형 이진 검색 트리
 - 검색, 삽입, 삭제 시 최악의 경우 $O(\log N)$
 - AVL 트리, RB 트리
- 균형 다진 검색 트리
 - 검색, 삽입, 삭제 시 최악의 경우 $O(\log N)$
 - 2-3 트리, 2-3-4 트리, B-트리
- 해시 테이블
 - 검색, 삽입, 삭제 시 평균 $O(1)$



시간 복잡도와 공간 복잡도 (3/3)

- 알고리즘: 시간 복잡도와 공간 복잡도

- (배열) 정렬 알고리즘

| 알고리즘 | 시간 복잡도 | | | 공간 복잡도 |
|-------|---------------|---------------|---------------|-------------|
| | 최악 | 평균 | 최선 | |
| 선택 정렬 | $O(N^2)$ | $O(N^2)$ | $O(N^2)$ | $O(1)$ |
| 버블 정렬 | $O(N^2)$ | $O(N^2)$ | $O(N^2)$ | $O(1)$ |
| 삽입 정렬 | $O(N^2)$ | $O(N^2)$ | $O(N)$ | $O(1)$ |
| 셸 정렬 | $O(N^2)$ | $O(N^{1.25})$ | $O(N^{1.25})$ | $O(1)$ |
| 퀵 정렬 | $O(N^2)$ | $O(N \log N)$ | $O(N \log N)$ | $O(\log N)$ |
| 병합 정렬 | $O(N \log N)$ | $O(N \log N)$ | $O(N \log N)$ | $O(N)$ |
| 힙 정렬 | $O(N \log N)$ | $O(N \log N)$ | $O(N \log N)$ | $O(1)$ |
| 계수 정렬 | $O(N+k)$ | $O(N+k)$ | $O(N+k)$ | $O(k)$ |
| 기수 정렬 | $O(Nk)$ | $O(Nk)$ | $O(Nk)$ | $O(N+k)$ |
| 버킷 정렬 | $O(N^2)$ | $O(N+k)$ | $O(N+k)$ | $O(N)$ |

재귀와 귀납적 사고



- 자료구조와 알고리즘

백문이불여일타(百聞而不如一打)

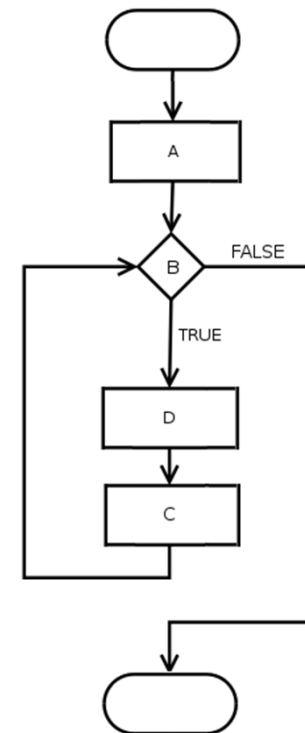
- 재귀와 귀납적 사고

- 재귀 함수, 재귀적.반복적 용법

- 피보나치 수열, 동적 프로그래밍

- 정렬과 검색 알고리즘

for(A;B;C)
D;



재귀 함수

● 재귀 함수(Recursive Function)

○ 자기 자신의 함수를 호출 함으로써, 반복적인 처리를 하는 함수

- 재귀 함수 안에서 사용하는 변수는 지역변수(자동변수)
- 재귀 함수의 인수들은 값에 의한 전달(pass by Value) 방식으로 전달된다.
- **주의: 반드시 탈출(종료) 조건 명시!!!**
 - **Stack Overflow 오류 발생 주의!!!**
- **장점**
 - 코드가 훨씬 간결 해지며, 프로그램을 보기가 쉽다.
 - 또한 프로그램 오류 수정이 용이하다.
- **단점**
 - 코드 자체를 이해하기 어렵다.
 - 또한 메모리 공간을 많이 요구한다.

재귀적.반복적 용법 (1/2)

● 재귀적 해법

- 큰 문제에 닮은 꼴의 작은 문제가 깃든다.
- 잘 쓰면 보약, 잘못 쓰면 맹독
 - 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다.
 - 재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다.
- 재귀적 해법이 바람직한 예
 - 계승(factorial) 구하기
 - 퀵 정렬, 병합 정렬 등의 정렬 알고리즘
 - 그래프의 깊이 우선 탐색(DFS, Depth First Search)
- 재귀적 해법이 치명적인 예
 - 피보나치 수 구하기
 - 행렬 곱셈 최적순서 구하기

재귀적.반복적 용법 (2/2)

● 계승(Factorial) 구하기

○ 반복적 정의

- 반복 함수가 반복적으로 정의된다.
 - 함수 정의는 매개변수를 포함하나 함수 자체는 포함하지 않는다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \dots 3 * 2 * 1 & \text{if } n > 0 \end{cases}$$

○ 재귀적 정의

- 함수가 자기 자신을 포함한다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

$\Theta(n)$

재귀와 귀납적 사고

재귀적.반복적 용법: 알고리즘 분석



재귀적.반복적 용법 (1/3)

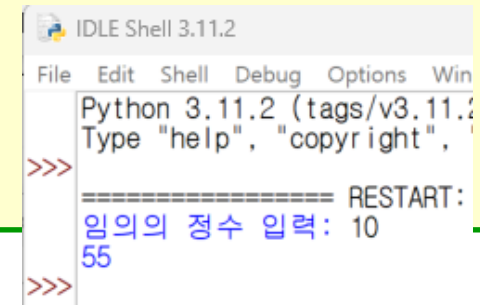
예제 1-1: 1부터 n까지의 합

| Python

재귀적 용법: 1부터 n까지의 합

```
def SUM(num:int) -> int:  
    if(num < 2): return 1  
    return num + SUM(num-1)
```

```
if __name__ == '__main__':  
    num = int(input('임의의 정수 입력: '))  
    print(f'{SUM(num)}', end='')
```



```
IDLE Shell 3.11.2  
File Edit Shell Debug Options Win  
Python 3.11.2 (tags/v3.11.2  
Type "help", "copyright",  
>>> ===== RESTART:  
임의의 정수 입력: 10  
55  
>>>
```

반복적 용법: 1부터 n까지의 합

```
def SUM(num:int) -> int:  
    tot = 0  
    for i in range(1, num+1):  
        tot += i
```

```
    return tot          # return sum(range(1, num+1))      # O(n)
```

```
    # return num * (num+1) // 2                             # O(1)
```

알고리즘 분석

$$sum(n) = 1 + 2 + 3 + \dots + (n - 1) + n$$

재귀적.반복적 용법 (2/3)

예제 1-1: 1부터 n까지의 합

| C++

```
#include <iostream>
using namespace std;
int SUM(int num);
int main(void)
{
    int    num;

    cout << "임의의 정수 입력: ";
    cin >> num;

    cout << "1부터 " << num << "까지의 합: " << SUM(num) << endl;
    return 0;
}
```

Microsoft Visual Studio 디버그 x + v

임의의 정수 입력: 10
1부터 10까지의 합: 55

C:\Users\click\OneDrive\문서\cppClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...

```
// 재귀적 용법: 1부터 n까지의 합
int SUM(int num) {
    if (num < 0)
        return 0;
    return num + SUM(num - 1);
}
```

알고리즘 분석

$$sum(n) = 1 + 2 + 3 + \dots + (n - 1) + n$$

```
// 반복적 용법: 1부터 n까지의 합
int SUM(int num) {
    int    tot = 0;
    for (int i = 1; i < num + 1; i++)
        tot += i;

    return tot;
    // return num * (num+1) / 2
}
```

O(n)
O(1)

재귀적.반복적 용법 (3/3)

예제 1-1: 1부터 n까지의 합

| C

```
#include <stdio.h>
int SUM(int num);
int main(void)
{
    int    num;

    printf("임의의 정수 입력: ");
    scanf_s("%d", &num);           // scanf("%d", &num);

    printf("1부터 %d까지의 합: %d \n", num, SUM(num));
    return 0;
}
```

Microsoft Visual Studio 디버그 × + ▾

임의의 정수 입력: 10
1부터 10까지의 합: 55

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...

```
// 재귀적 용법: 1부터 n까지의 합
int SUM(int num) {
    if (num < 0)
        return 0;
    return num + SUM(num - 1);
}
```

알고리즘 분석

$$sum(n) = 1 + 2 + 3 + \dots + (n - 1) + n$$

```
// 반복적 용법: 1부터 n까지의 합
int SUM(int num) {
    int    tot = 0;
    for (int i = 1; i < num + 1; i++)
        tot += i;

    return tot;
    // return num * (num+1) / 2
}
```

O(n)
O(1)

재귀와 귀납적 사고

피보나치 수열

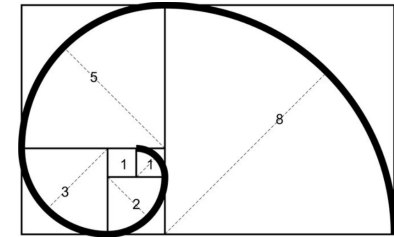


피보나치 수열

● 피보나치 수열(Fibonacci Sequence)

○ 피보나치(Fibonacci)

- 1,200년 경에 활동한 이탈리아 수학자



“토끼 한 마리가 매년 새끼 한 마리를 낳는다.
새끼는 한 달 후부터 새끼를 낳기 시작한다.
최초 토끼 한 마리가 있다고 하면...
한 달 후에 토끼는 두 마리가 되고 두 달 후에는 세 마리가 되고...”

// 재귀적 용법: 피보나치 수열

Fibonacci(num)

{

if (num = 1 or num = 2)
then return 1;

else

return (**Fibonacci**(num - 1) + **Fibonacci**(num - 2));

}

$$f_n = f_{n-1} + f_{n-2} (n \geq 3)$$

$$f_1 = f_2 = 1 (n = 1, 2)$$

“아주 간단한 문제지만...
동적 프로그래밍의 동기와 구현이 다 포함되어 있다.”

피보나치 수열: 동적 프로그래밍 (1/2)

- 동적 프로그래밍의 적용 조건

- 최적 부분 구조(Optimal Substructure)

- 큰 문제의 해답에 그보다 작은 문제의 해답이 포함되어 있다.
 - 최적 부분 구조를 가진 문제의 경우에는 재귀 호출을 이용하여 문제를 풀 수 있다.

- 재귀 호출 시 중복(overlapping recursive calls)

- 재귀적으로 구현했을 때 중복 호출로 심각한 비효율이 발생한다.

동적 프로그래밍이 그 해결책 !!!

- 위의 두 성질이 있는 문제에 대해 적절한 저장 방법으로 중복 호출의 비효율을 제거한 것을 동적 프로그래밍이라고 한다.

피보나치 수열: 동적 프로그래밍 (2/2)

- 피보나치 수열: 동적 프로그래밍

Fibonacci(n)

```
{  
    f[1] ← f[2] ← 1;  
    for i ← 3 to n  
        f[i] ← f[i - 1] + f[i - 2];  
  
    return f[n];  
}
```

fibonacci

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | ? | ? | ? | ? | ? |
|---|---|---|---|---|---|---|

$\Theta(n)$

Fibonacci(n)

```
{  
    first ← second ← 1;  
    for i ← 3 to n  
        res ← first + second;  
    return res;  
}
```

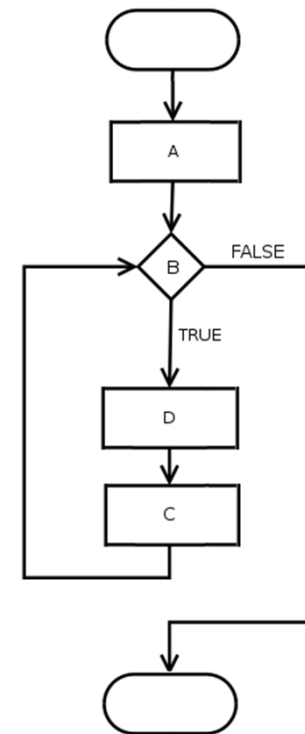
정렬과 검색 알고리즘



백문이불여일타(百聞而不如一打)

- 자료구조와 알고리즘
- 재귀와 귀납적 사고
- 정렬과 검색 알고리즘
 - 정렬과 검색 알고리즘

for(A;B;C)
D;



기초적인 정렬과 검색 알고리즘

선택.버블.삽입 정렬



선택 정렬

선택 정렬 동작 과정



수행시간: $(n - 1) + (n - 2) + \dots + 2 + 1 = O(N^2)$

Worst case

Average case

버블 정렬

버블 정렬 동작 과정

초기 배열

| | | | | |
|----|----|----|----|----|
| 30 | 40 | 10 | 50 | 20 |
|----|----|----|----|----|

| | | | | |
|----|----|----|----|----|
| 30 | 40 | 10 | 20 | 50 |
|----|----|----|----|----|

| | | | | |
|----|----|----|----|----|
| 30 | 40 | 10 | 20 | 50 |
|----|----|----|----|----|

| | | | | |
|----|----|----|----|----|
| 30 | 10 | 40 | 20 | 50 |
|----|----|----|----|----|

1단계 완료 이후

| | | | | |
|----|----|----|----|----|
| 10 | 30 | 40 | 20 | 50 |
|----|----|----|----|----|

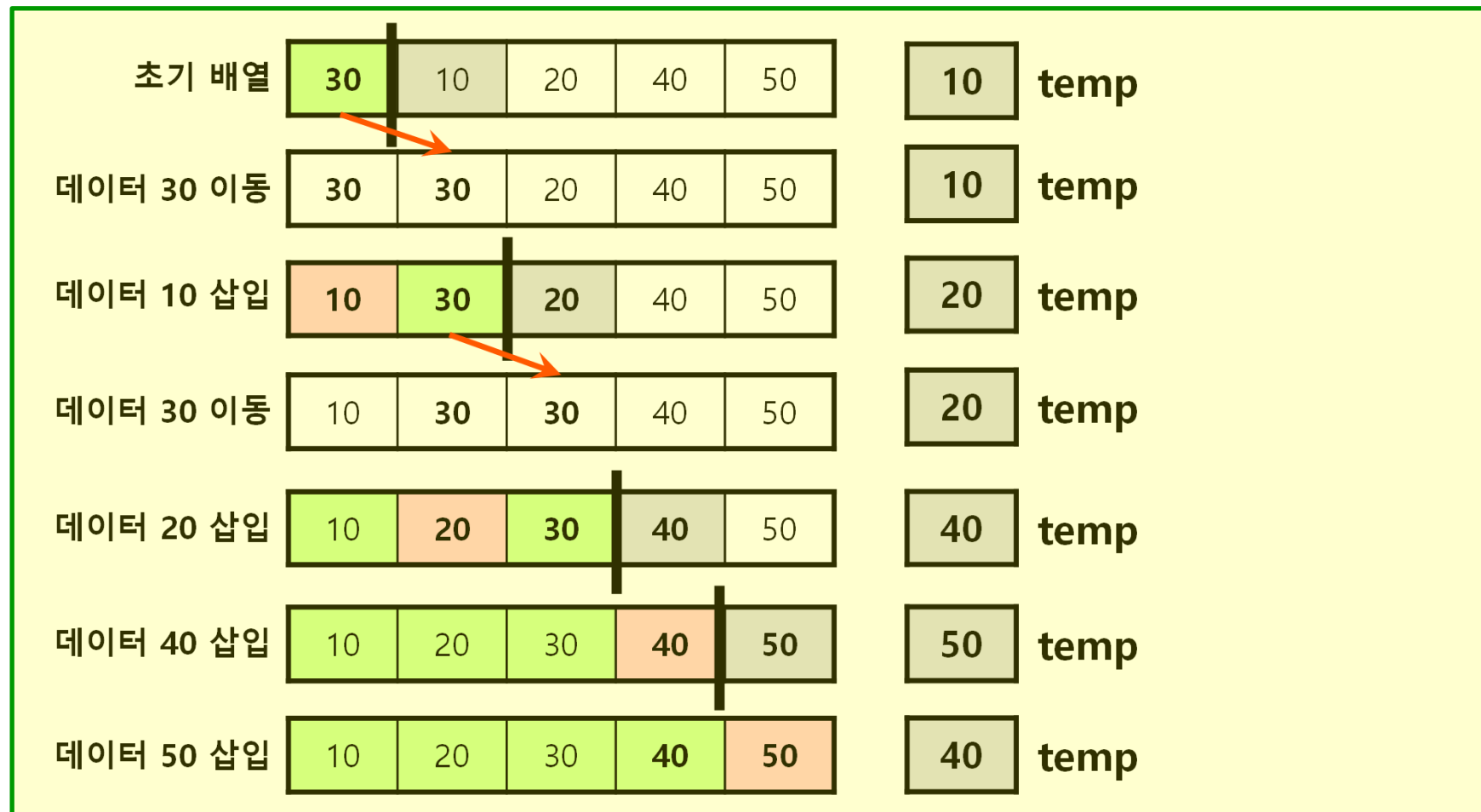
수행시간: $1 + 2 + \dots + (n - 1) + n = O(N^2)$

Worst case

Average case

삽입 정렬 동작 과정

삽입 정렬



수행시간: $O(N^2)$

Worst case: $1 + 2 + \dots + (n-2) + (n-1)$

Average case: $\frac{1}{2}(1 + 2 + \dots + (n-2) + (n-1))$

기초적인 정렬과 검색 알고리즘

순차.이진 검색



순차 검색 (1/2)

- 순차 검색: 동작 과정 -- 검색 성공

목표 데이터: 73

- 순서 없는 리스트에 위치한 데이터

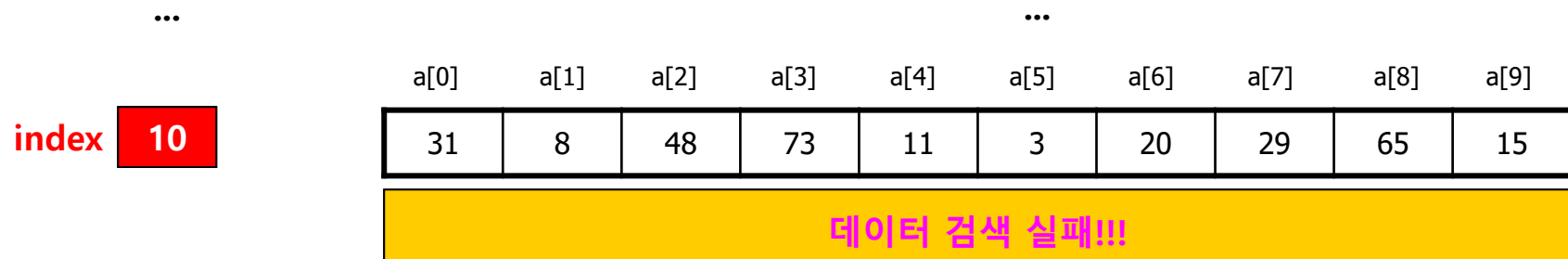
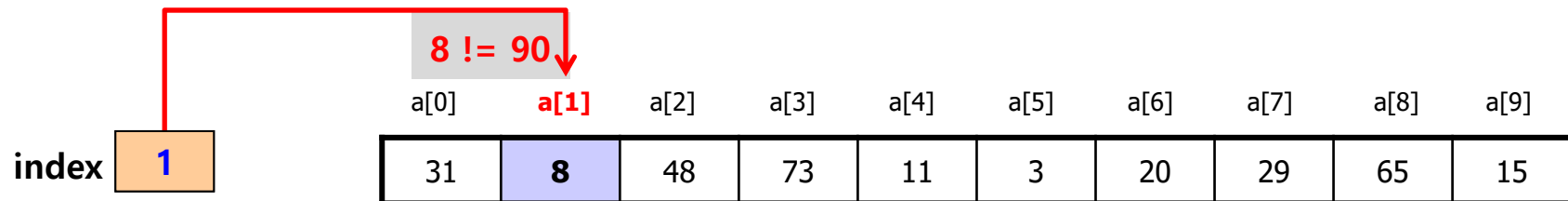
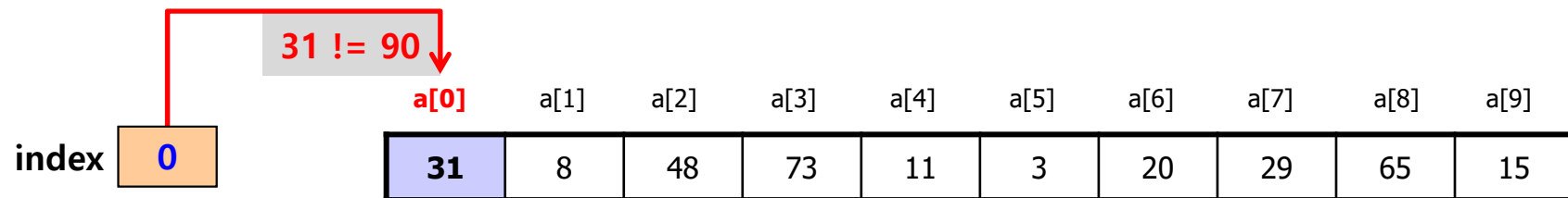


순차 검색 (2/2)

- 순차 검색: 동작 과정 -- 검색 실패

목표 데이터: 90

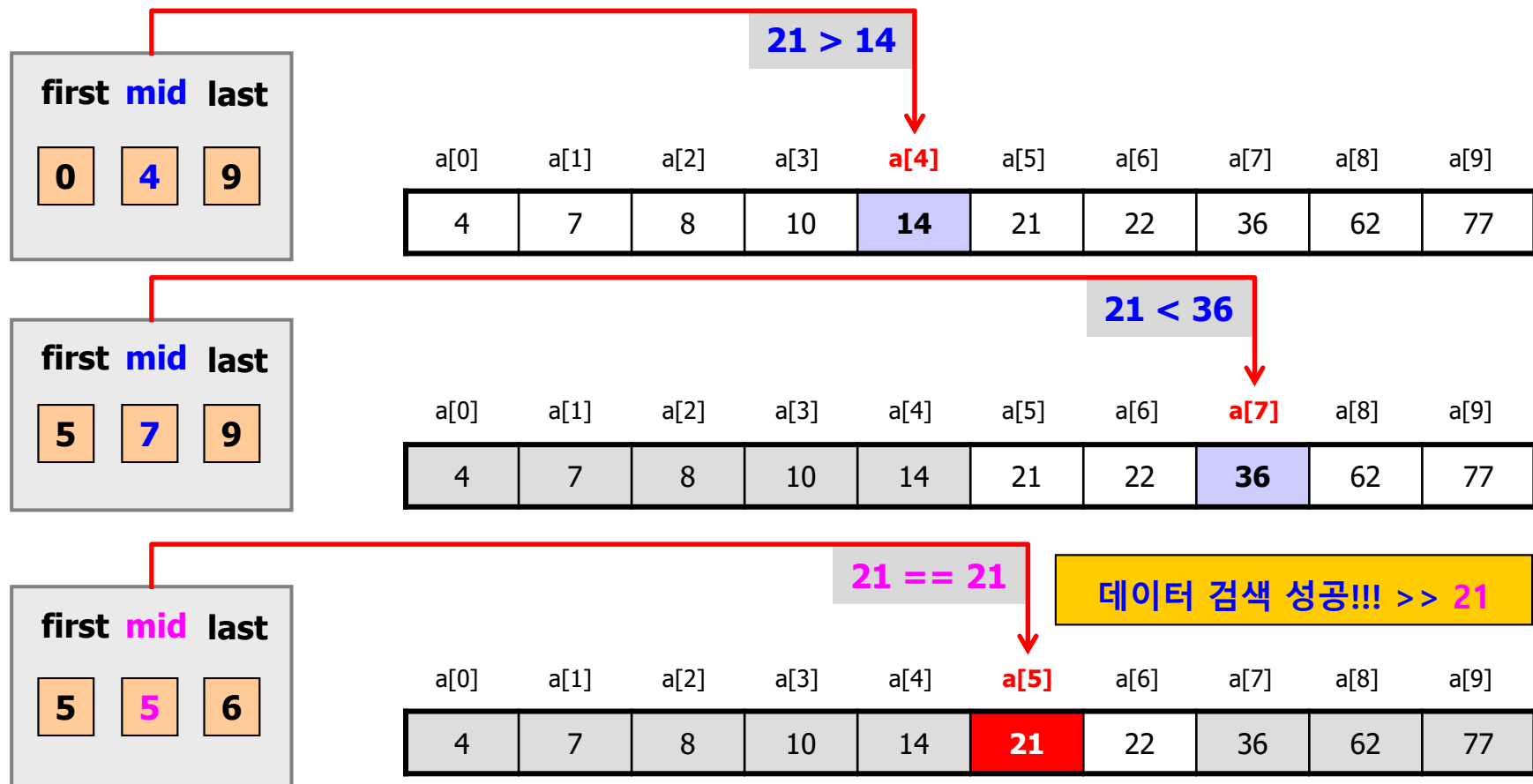
- 순서 없는 리스트에 검색 실패



이진 검색 (1/2)

- 이진 검색: 동작 과정 -- 검색 성공

검색 데이터: 21



이진 검색 (2/2)

● 이진 검색: 동작 과정 -- 검색 실패

검색 데이터: 11

| first | mid | last |
|-------|-----|------|
| 0 | 4 | 9 |

11 < 14

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] |
|------|------|------|------|------|------|------|------|------|------|
| 4 | 7 | 8 | 10 | 14 | 21 | 22 | 36 | 62 | 77 |

| first | mid | last |
|-------|-----|------|
| 0 | 1 | 3 |

11 > 7

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] |
|------|------|------|------|------|------|------|------|------|------|
| 4 | 7 | 8 | 10 | 14 | 21 | 22 | 36 | 62 | 77 |

| first | mid | last |
|-------|-----|------|
| 2 | 2 | 3 |

11 > 8

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] |
|------|------|------|------|------|------|------|------|------|------|
| 4 | 7 | 8 | 10 | 14 | 21 | 22 | 36 | 62 | 77 |

| first | mid | last |
|-------|-----|------|
| 3 | 3 | 3 |

11 > 10

| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] | a[9] |
|------|------|------|------|------|------|------|------|------|------|
| 4 | 7 | 8 | 10 | 14 | 21 | 22 | 36 | 62 | 77 |

| first | mid | last |
|-------|-----|------|
| 4 | 3 | 3 |

first > last

데이터 검색 실패!!!

고급 정렬 알고리즘

셀.퀵.병합 정렬



셸 정렬

- 셸 정렬(Shell Sort)

- 일정한 간격(interval)으로 데이터들끼리 부분집합을 구성하고, 각 부분집합에 있는 원소들에 대해서 삽입 정렬을 수행한다.

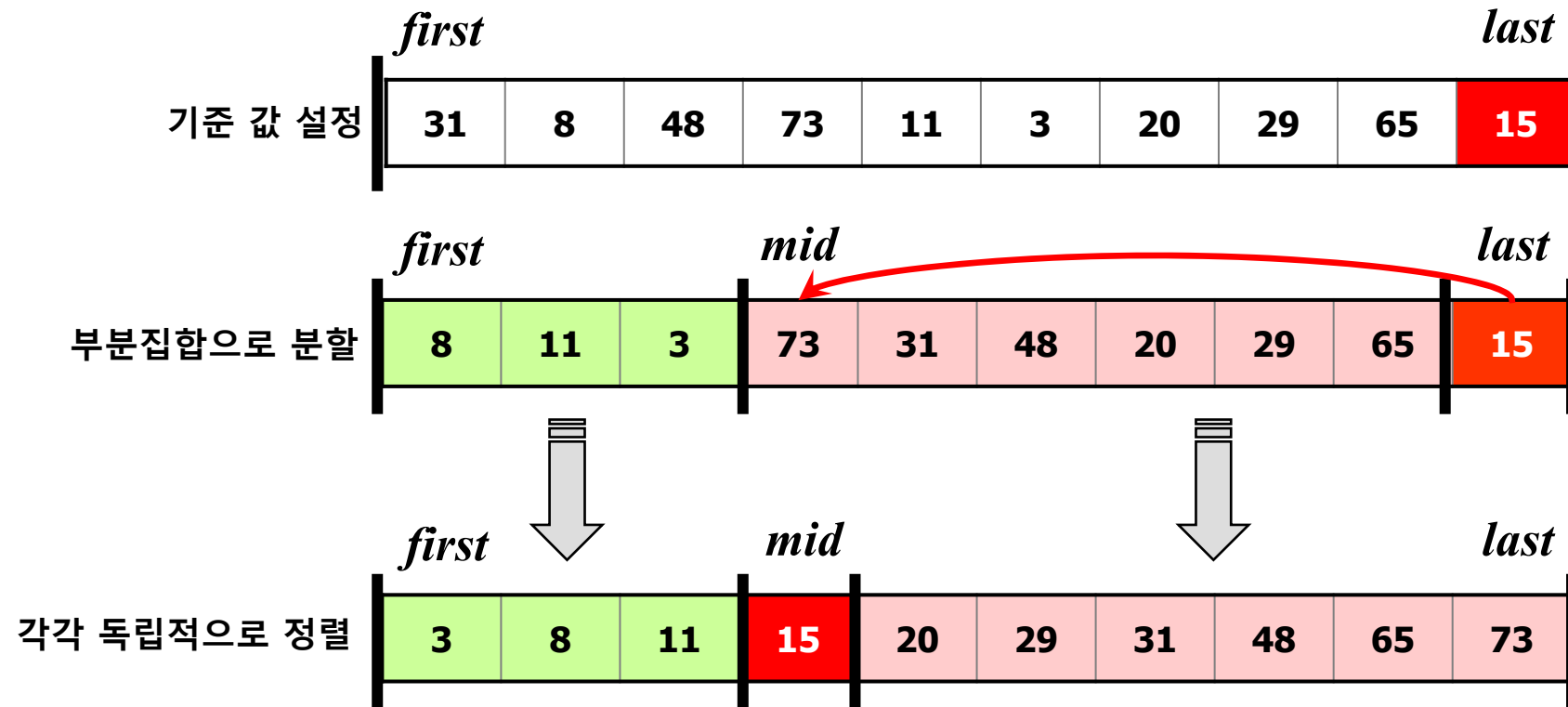
- 전체 원소에 대해서 삽입 정렬을 수행하는 것보다 부분집합으로 나누어 정렬하면 **비교와 교환 연산을 감소**시킬 수 있다.
- 셸 정렬에서는 7-정렬, 4-정렬 등의 용어를 주로 사용

- 4-정렬의 예

| | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 30 | 75 | 15 | 40 | 10 | 65 | 35 | 20 | 90 | 55 | 95 | 25 |
|----|----|----|----|----|----|----|----|----|----|----|----|

퀵 정렬

- 퀵 정렬: 동작 과정

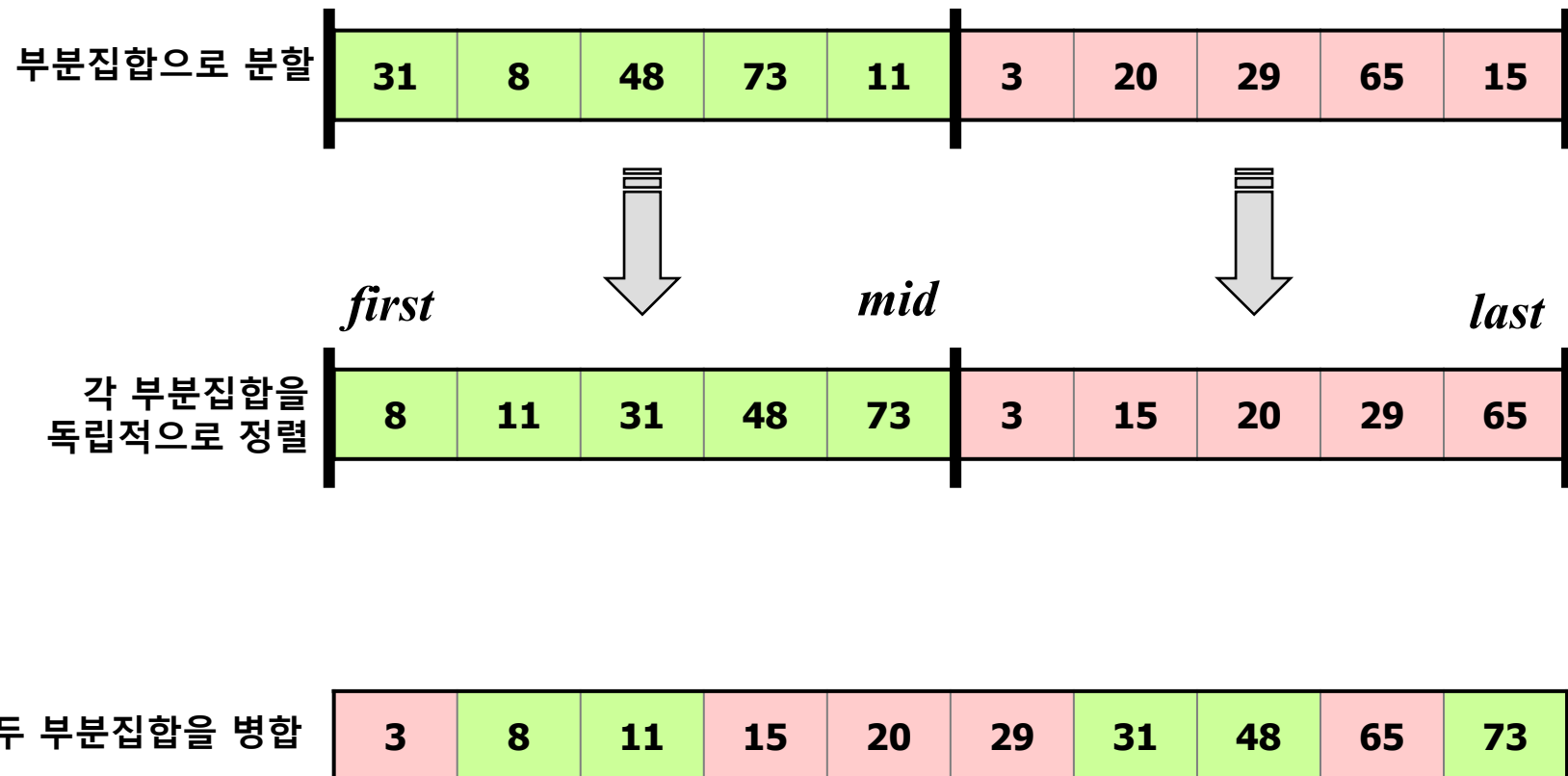


평균 수행시간: $O(N\log N)$

최악의 경우 수행시간: $O(N^2)$

병합 정렬

- 병합 정렬: 동작 과정



특수 정렬 알고리즘

계수.기수.버킷 정렬



특수 정렬 알고리즘

- 특수 정렬 알고리즘

- 비교 정렬

- 두 원소를 비교하는 정렬의 하한선은 $\Omega(N \log N)$

“최악의 경우 정렬 시간이 $O(N \log N)$ 보다 더 빠를 수는 없는가?”

- 그러나 원소들이 특수한 성질을 만족하면 $O(n)$ 정렬도 가능하다.
 - 계수 정렬(Counting Sort): 원소들의 크기가 모두 $-O(N) \sim O(N)$ 범위에 있을 때...
 - 기수 정렬(Radix Sort): 원소들이 모두 k 이하의 자리 수를 가졌을 때(k : 상수)
 - 버킷 정렬(Bucket Sort): 원소들이 균등 분포(Uniform distribution)를 이룰 때...

계수 정렬

- **계수 정렬: 동작과정**

- **1단계**

- ① data에서 각 항목들의 발생 횟수를 센다.
- ② 발생 횟수들은 정수 항목들로 직접 인덱스 되는 카운트 배열(counts)에 저장한다.

처음의 정렬되지 않은 집합

data

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 4 | 1 | 3 | 1 | 2 | 4 | 1 |
|---|---|---|---|---|---|---|---|

data의 각 정수의 발생 횟수

counts

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 1 | 1 | 2 |
|---|---|---|---|---|

counts[0] counts[1] counts[2] counts[3] counts[4]

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 5 | 6 | 8 |
|---|---|---|---|---|

counts

counts[4] = 4의 발생 횟수
counts[3] = 3의 발생 횟수
counts[2] = 2의 발생 횟수
counts[1] = 1의 발생 횟수
counts[0] = 0의 발생 횟수

- ③ 정렬된 집합에서 각 항목의 앞에 위치할 항목의 개수를 반영하기 위하여 카운트들을 조정한다.

- **2단계: 정렬된 집합**

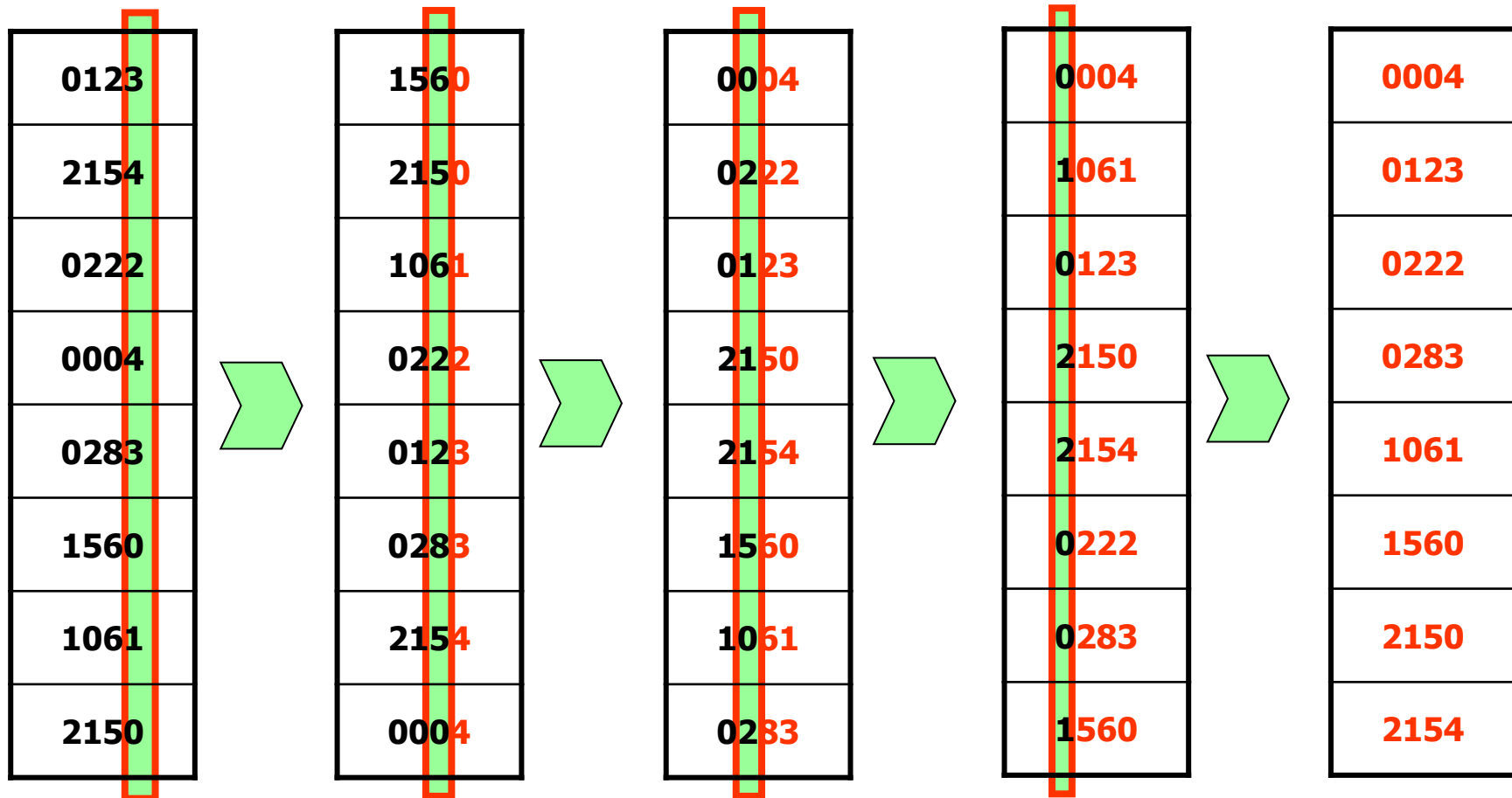
temp

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 1 | 1 | 1 | 2 | 3 | 4 | 4 |
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] |

47

기수 정렬

- 기수 정렬: 동작 과정

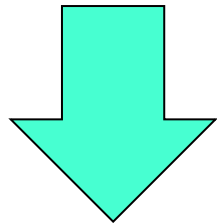


버킷 정렬 (1/2)

- 버킷 정렬: 동작 과정

(a) $A[0...14]$: 정렬할 배열

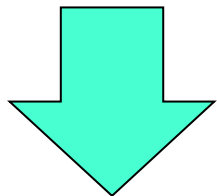
| | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| .38 | .94 | .48 | .73 | .99 | .43 | .55 | .15 | .85 | .84 | .81 | .71 | .17 | .10 | .02 |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|



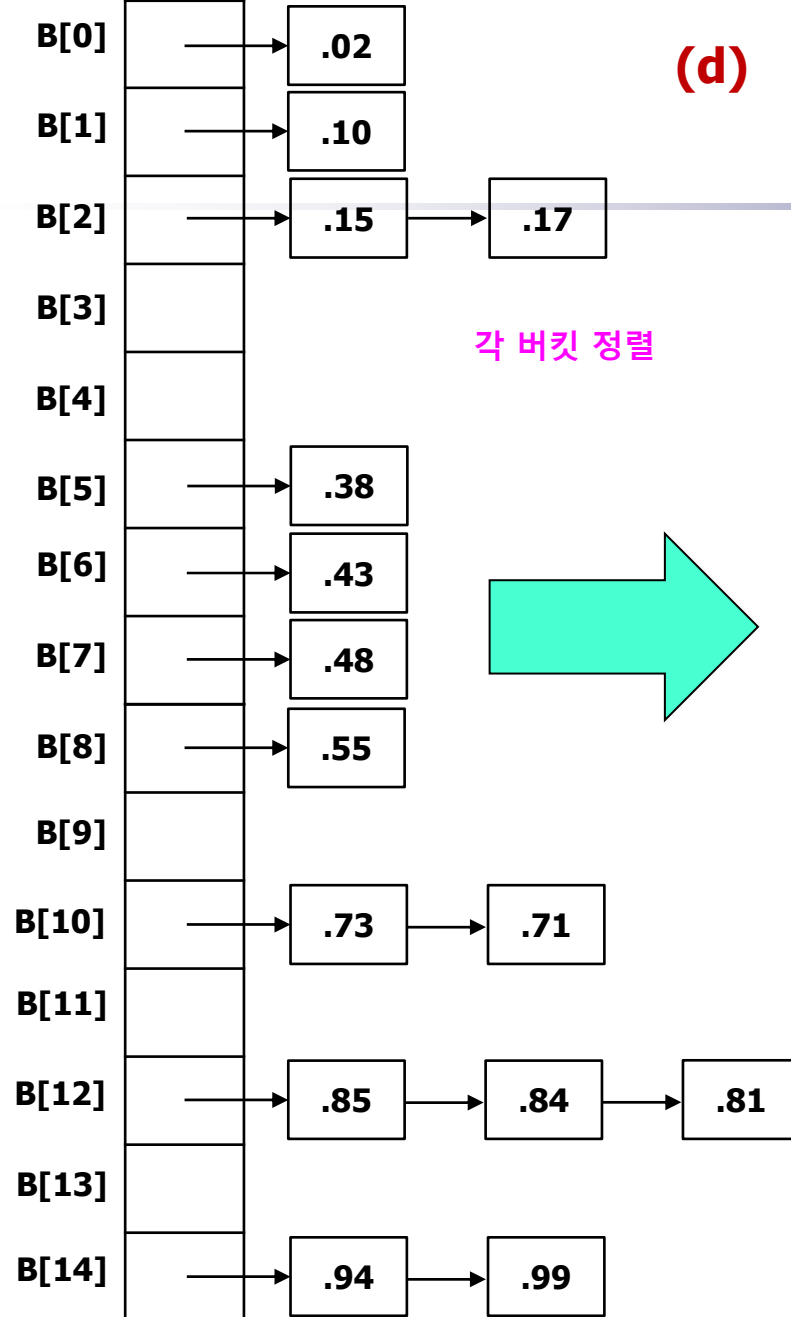
$A[0...14]$ 각각에 15를 곱하여 정수부만 취함.

(b) 버킷 리스트 위치

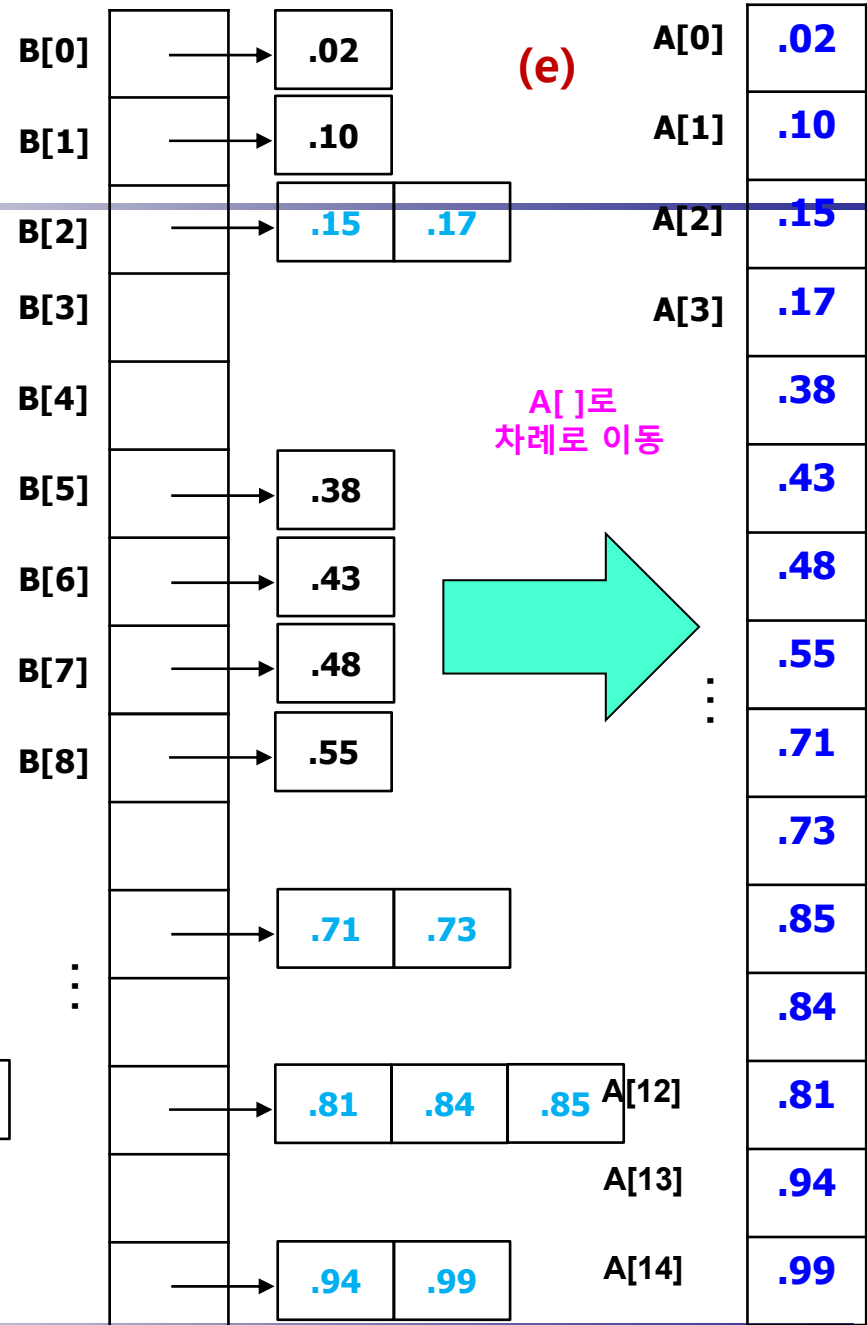
| | | | | | | | | | | | | | | |
|---|----|---|----|----|---|---|---|----|----|----|----|---|---|---|
| 5 | 14 | 7 | 10 | 14 | 6 | 8 | 2 | 12 | 12 | 12 | 10 | 2 | 1 | 0 |
|---|----|---|----|----|---|---|---|----|----|----|----|---|---|---|



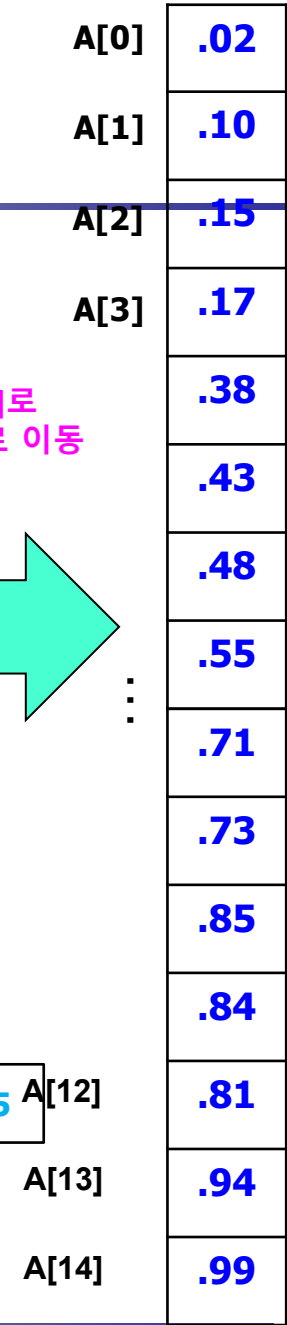
(c)



(d)

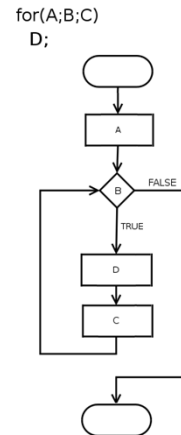


(e)



참고문헌

- [1] "IT CookBook, 쉽게 배우는 자료구조 with 파이썬", 문병로, 한빛아카데미, 2022.
- [2] "이것이 자료구조+알고리즘이다: with C 언어", 박상현, 한빛아카데미, 2022.
- [3] "C++로 구현하는 자료구조와 알고리즘(2판)", Michael T. Goodrich, 김유성 외 2인 번역, 한빛아카데미, 2020.
- [4] 문병로, "IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법"(개정판), 개정판, 한빛아카데미, 2018.
- [5] "이것이 취업을 위한 코딩 테스트다 with 파이썬", 나동빈, 한빛미디어, 2020.
- [6] "코딩 테스트를 위한 자료 구조와 알고리즘 with C++", John Carey 외 2인, 황선규 역, 길벗, 2020.
- [7] "SW Expert Academy", SAMSUNG, 2024 of viewing the site, <https://swexpertacademy.com/>.
- [8] "BAEKJOON", (BOJ) BaekJoon Online Judge, 2024 of viewing the site, <https://www.acmicpc.net/>.
- [9] "programmers", grepp, 2024 of viewing the site, <https://programmers.co.kr/>.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며,
내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.

