

# 알고리즘의 성능 분석

## ▶ 알고리즘의 성능 분석 기법

### 실행 시간을 측정하는 방법

- 두 개의 알고리즘의 실제 실행 시간을 측정하는 것
- 실제로 구현하는 것이 필요
- 동일한 하드웨어를 사용하여야 함

### 알고리즘의 복잡도를 분석하는 방법

- **시간 복잡도 분석** : 수행 시간 분석
- **공간 복잡도 분석** : 수행시 필요로 하는 메모리 공간 분석
- 직접 구현하지 않고서도 수행 시간을 분석하는 것
- 알고리즘이 수행하는 연산의 횟수를 측정하여 비교
- 일반적으로 연산의 횟수는  $n$ 의 함수

# 시간 복잡도 분석

## ▶ 시간 복잡도 분석

산술, 대입, 비교, 이동의 기본적인 연산 고려

- 알고리즘 수행에 필요한 연산의 개수를 계산
- 입력의 개수  $n$ 에 대한 함수->**시간 복잡도 함수**

big-O표기법 :

알고리즘 수행 단계를 바탕으로

포괄적으로 알고리즘성능의 범주를 정한 것

$O(1)$	상수시간
$O(N)$	선형시간
$O(N^2)$	이차시간
$O(\log N)$	로그시간

# 시간 복잡도 분석의 예

## n을 n번 더하는 문제

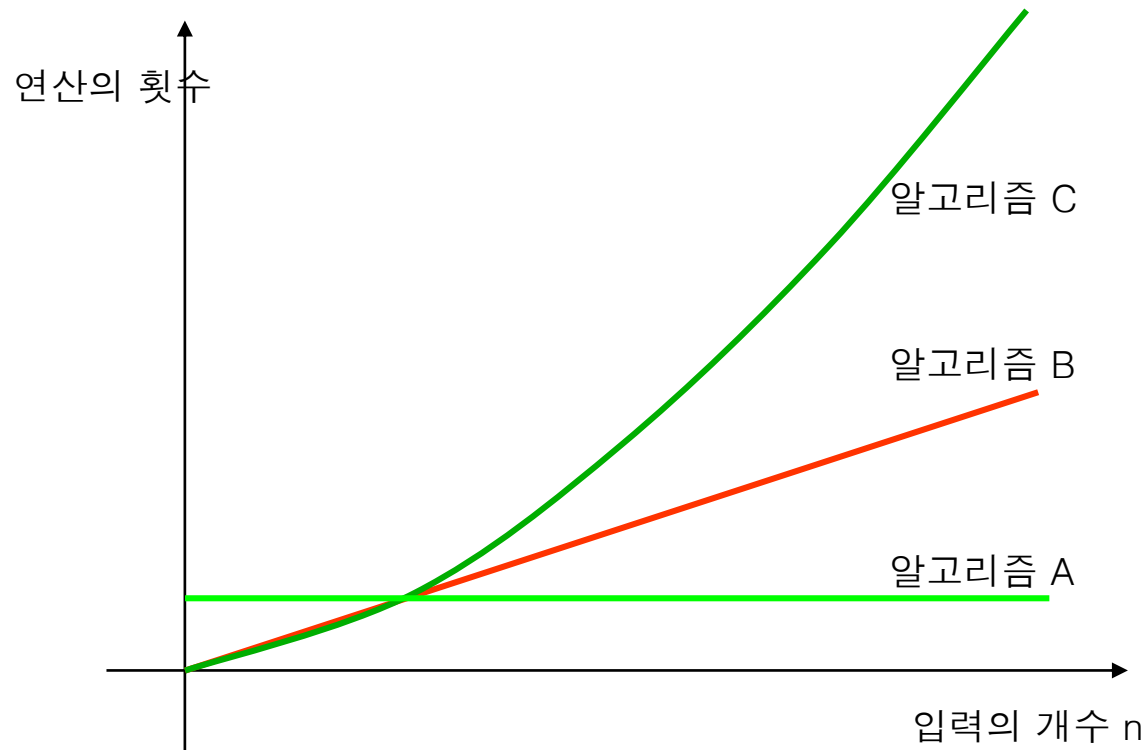
- 각 알고리즘이 수행하는 연산의 개수 계산
- 단 for 루프 제어 연산은 고려하지 않음

알고리즘 A	알고리즘 B	알고리즘 C
$\text{sum} \leftarrow n * n;$	$\text{sum} \leftarrow 0;$ $\text{for } i \leftarrow 1 \text{ to } n \text{ do}$ $\quad \text{sum} \leftarrow \text{sum} + n;$	$\text{sum} \leftarrow 0;$ $\text{for } i \leftarrow 1 \text{ to } n \text{ do}$ $\quad \text{for } j \leftarrow 1 \text{ to } n \text{ do}$ $\quad \quad \text{sum} \leftarrow \text{sum} + 1;$

	알고리즘 A	알고리즘 B	알고리즘 C
대입연산	1	$n + 1$	$n * n + 1$
덧셈연산		$n$	$n * n$
곱셈연산	1		
나눗셈연산			
전체연산수	2	$2n + 1$	$2n^2 + 1$

# 알고리즘의 성능분석

## ▶ 입력 크기에 대한 연산(단계)의 횟수를 그래프로 표현



# 시간복잡도의 계산

## ▶ 시간 복잡도 계산의 예

코드를 분석해보면 수행되는 수행 되는  
연산들의 횟수를 입력 크기의 함수로 만들 수 있다.

- best case
- average case
- worst case

```
double average(int a[], int n)
{
    int k;
    double sum;
    sum = 0;
    for (k=1; k < n; k++)
        sum = sum + a[k];
    return sum/n;
}
```

```
int find_max(int a[], int n)
{
    int k, max_data;
    max_data = a[0];
    for (k=1; k < n; k++)
        if (a[k] > max_data)
            max_data = a[k];
    return max_data;
}
```

98 90 80 70 60 50

50 60 70 80 90 98

# 시간복잡도의 계산

## ▶ 시간 복잡도 계산의 예 (순차검색~이진검색)

코드를 분석해보면 수행되는

연산들의 횟수를 입력 크기의 함수로 만들 수 있다.

- best case
- average case
- worst case

```
int ssearch(int a[], int n int sdata)
{
    int k;

    for (k=0; k < n; k++)
        if (a[k] == sdata)
            return k;
    return -1;
}
```

# 시간복잡도의 계산

## ▶ 시간 복잡도 계산의 예 (순차검색~이진검색)

코드를 분석해보면 수행되는 수행 되는

연산들의 횟수를 입력 크기의 함수로 만들 수 있다.

```
int bsearch(int a[], int n, int key)
{
    int mid;
    int left = 0, right = n-1;
    while (left <= right) {
        mid = (left + right) / 2;
        if (key > a[mid]) left = mid + 1;
        else if (key < a[mid]) right = mid - 1;
        else return mid;
    } /* while */
    return -1;
}
```

원소갯수	O(n)	O(logn)
8	8	3
16	16	4
32	32	5
64	64	6
128	128	7
256	256	8
512	512	9
1024	1024	10

# 알고리즘 성능 분석

## ▶ 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	$26313 \times 10^{33}$



# 알고리즘 성능 분석

## ▶ 빅오 표기법의 종류

$O(1)$  : 상수형

$O(\log n)$  : 로그형

$O(n)$  : 선형

$O(n \log n)$  : 로그선형

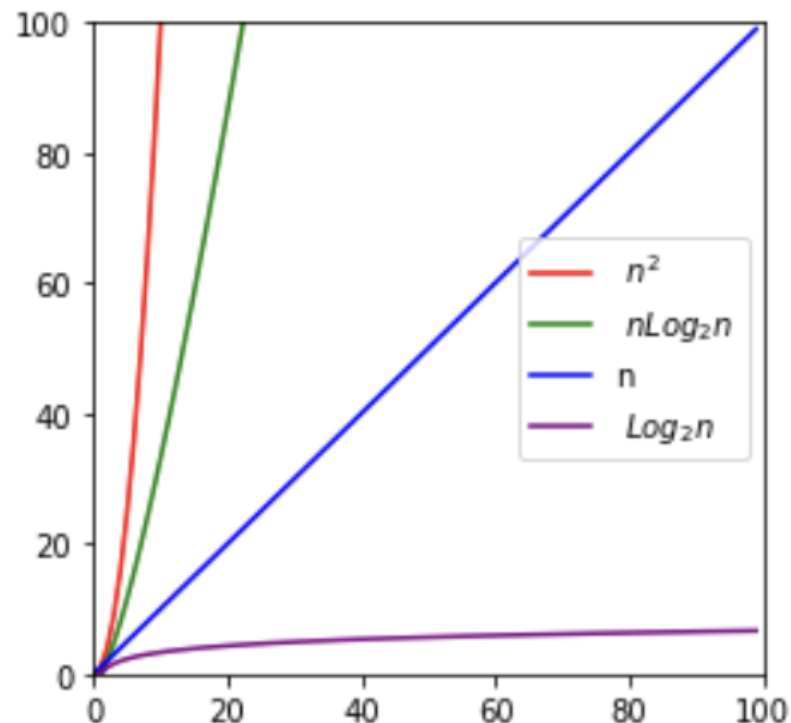
$O(n^2)$  : 2차형

$O(n^3)$  : 3차형

$O(n^k)$  : k차형

$O(2^n)$  : 지수형

$O(n!)$  : 팩토리얼형



# 정렬 알고리즘 설계 및 구현

---

## 1. 개요

정보처리에 있어 정렬의 중요성

기준 값(key)이 커지는 순으로(오름차순)

또는 작아지는 순으로(내림차순) 데이터를 나열하는 것

## 2. 버블 정렬(Bubble Sort)

## 3. 삽입 정렬(Insertion Sort)

## 4. 퀵 정렬(Quick Sort)

## 5. 이진합병 정렬(2-way Merge Sort)

---

# 버블 정렬(Bubble Sort)

## 1. 기본 방법

인접한 두 자료( $a[i]$ 와  $a[i+1]$ )를 비교하여 오름차순으로 저장되어 있지 않으면 교환

```
void bubble(int a[], int n)
{
    int i=n-1, j, tmp;

    while (i != 0) {
        for (j=0; j <= i-1; j++) {
            if (a[j] > a[j+1]) {
                tmp = a[j];
                a[j] = a[j+1];
                a[j+1] = tmp;
            }
        }
        i--;
    }
}
```

n=7	0	1	2	3	4	5	6
a	25	48	37	17	52	86	43
i=6	25	37	17	48	52	43	86
i=5	25	17	37	48	43	52	86
i=4	17	25	37	43	48	52	86
i=3							
i=2							
i=1							

# 버블 정렬(Bubble Sort)

## 2. 개선된 방법

한번의 pass동안 한번도 교환이 이루어지지 않으면 끝내도록 개선  
flag변수의 사용

```
void bubble(int a[], int n)
{
    int i=n-1, j, tmp, flag = 1 ;

    while ( flag && i != 0) {
        flag = 0;
        for (j=0; j <= i-1; j++) {
            if (a[j] > a[j+1]) {
                flag = 1;
                tmp = a[j]; a[j] = a[j+1]; a[j+1] = tmp;
            }
        }
        i--;
    }
}
```

# 버블정렬 정리 및 실습

1. 다음 오름차순으로 정렬하는 버블정렬 알고리즘을 분석하면서 다음 예에 대하여 실행 한 후 출력되는 것을 쓰시오. 또한 내림차순의 경우 무엇을 바꾸어야하는지 코드를 쓰시오

```
a={12, 17, 25, 33, 48, 52, 86, 37, 100, 70}
Void bubble_flag(int a[], int n)
{
    int i=n-1, j, tmp, flag = 1;
    while (flag && i != 0) {
        flag = 0;
        for (j=0; j <= i-1; j++) {
            if (a[j] > a[j+1]) {
                flag = 1;
                tmp = a[j]; a[j] = a[j+1]; a[j+1] = tmp
            } //if
        } //for
        i--;
    }
    printf("i = %d :: flag = %d\n", i, flag);
}
```

```
main()
{
    int list[]={12, 17, 25, 33, 48, 52, 86, 37, 100, 70};
    int i, n;

    n=sizeof(list)/sizeof(int);
    bubble(list, n);
    printf("정렬된 데이터 리스트: \n");
    for (i=0; i < n; i++)
        printf("%d ", list[i]);
}
```

# 버블정렬 정리 및 실습

2. 다음에서 설명하는 함수 `bubble()`, `print_item()`, `bsearch_stock()`을 작성하고 이를 호출하는 `main()`을 작성하여 완성하는 과정을 이해하시오

각 행에 물품코드와 그 물품의 재고개수를 임의의 순으로 저장하고 있는 파일(`bitem.d`)로부터

$n \times 2$ 의 2차원배열 `stock`에 데이터를 읽어 들인다.

`bubble()` 정렬에 의하여 물품의 내림차순으로 정렬한 후

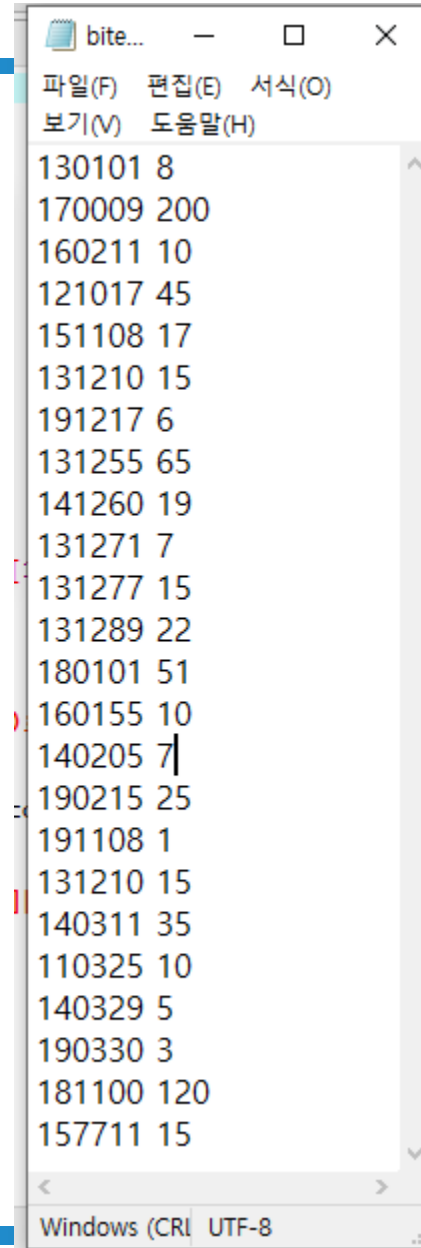
`print_item()`으로 `stock`에 저장하고 있는 물품코드를 출력해준다.

그리고 재고갯수를 확인하고자하는 물품코드(`item_code`)를 입력받아 이를 파라미터로 함수 `bsearch_stock()`를 호출한다.

`bsearch_stock`함수는 이진검색을 한 후,

그 물품이 있으면 해당 물품의 재고갯수를,

없으면 재고물품이 없다고 출력한다.



```
bite...
파일(F) 편집(E) 서식(O)
보기(V) 도움말(H)
130101 8
170009 200
160211 10
121017 45
151108 17
131210 15
191217 6
131255 65
141260 19
131271 7
131277 15
131289 22
180101 51
160155 10
140205 7
190215 25
191108 1
131210 15
140311 35
110325 10
140329 5
190330 3
181100 120
157711 15
Windows (CRLF) UTF-8
```



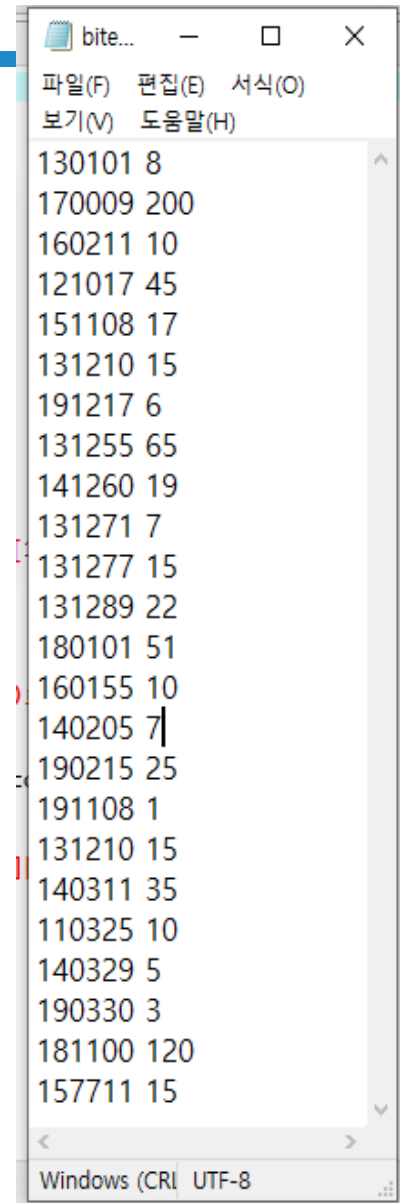
# 버블정렬 정리 및 실습

```
int bsearch_stock(unsigned a[][2], int n, unsigned key);
void print_item(char *heading, unsigned a[][2], int n);
void bubble(unsigned a[][2], int n);
main(int argc, char *argv[])
{
    FILE *stockdb;
    unsigned stock[INUM][2], item_code;
    int k = 0, s;

    if ((stockdb = fopen(argv[1], "r")) == NULL) {
        printf("Cannot open read file.. ..\n"); exit(1);
    }
    while ((fscanf(stockdb, "%u %u", &stock[k][0], &stock[k][1])) != EOF)
        k++;
    print_item("임의로 저장된 물품코드 : \n", stock, k);

    //물품코드의 내림차순으로 정렬
    bubble(stock, k);
    print_item("내림차순으로 정렬된 물품코드 : \n", stock, k);

    // 검색
    printf("검색할 item number 입력 : "); scanf("%u", &item_code);
    s = bsearch_stock(stock, k, item_code);
    if (s == -1) printf("재고물품이 없습니다\n");
    else printf("%u의 재고개수 = %u\n", stock[s][0], stock[s][1]);
}
```



```
bite...
파일(F) 편집(E) 서식(O)
보기(V) 도움말(H)
130101 8
170009 200
160211 10
121017 45
151108 17
131210 15
191217 6
131255 65
141260 19
131271 7
131277 15
131289 22
180101 51
160155 10
140205 7
190215 25
191108 1
131210 15
140311 35
110325 10
140329 5
190330 3
181100 120
157711 15
Windows (CRLF) UTF-8
```



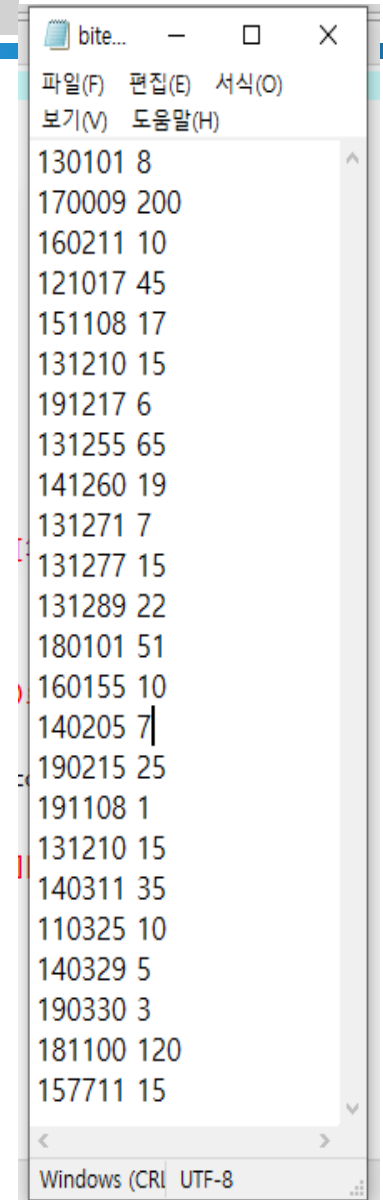
# 버블정렬 정리 및 실습

```
void bubble(unsigned a[][2], int n)
{
    int i = n - 1, j, flag = 1;
    unsigned tmp0, tmp1;

    while (flag && i != 0) {
        flag = 0;
        for (j = 0; j <= i - 1; j++) {

            if (a[j][0] < a[j+1][0]) {
                flag = 1;
                tmp0 = a[j][0]; a[j][0] = a[j+1][0]; a[j+1][0] = tmp0;
                tmp1 = a[j][1]; a[j][1] = a[j+1][1]; a[j+1][1] = tmp1;
            } //if

        } //for
        i--;
    }
}
```

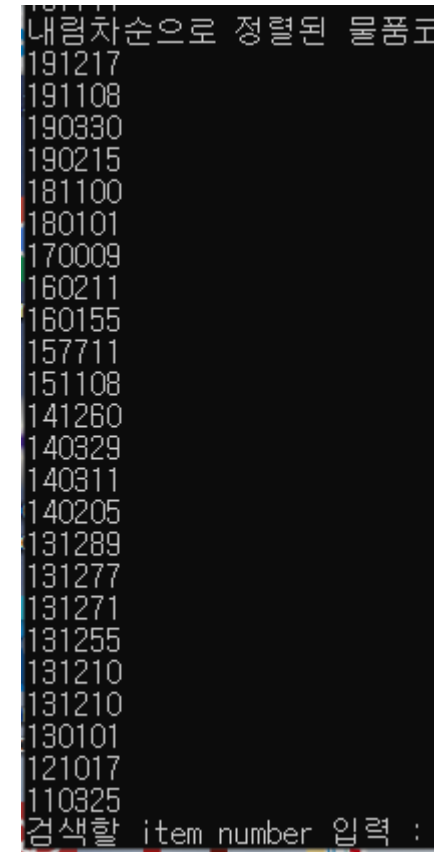


# 버블정렬 정리 및 실습

```
void print_item(char *heading, unsigned a[][2], int n)
{
    int i;
    printf(heading);
    for (i = 0; i < n; i++) printf("%u\n", a[i][0]);
}

int bsearch_stock(unsigned a[][2], int n, unsigned key)
{
    int left = 0, right = n - 1, mid;           //선언문

    while (left <= right) {
        mid = (left + right) / 2;
        if (key < a[mid][0]) left = mid + 1;
        else if (key > a[mid][0]) right = mid - 1;
        else return mid;           /* find */
    } /* while */
    return -1;    /* not exist */
} /* search */
```



내림차순으로 정렬된 물품코

191217  
191108  
190330  
190215  
181100  
180101  
170009  
160211  
160155  
157711  
151108  
141260  
140329  
140311  
140205  
131289  
131277  
131271  
131255  
131210  
131210  
130101  
121017  
110325

검색할 item number 입력 :

# 삽입 정렬(Insertion Sort)

## 기본 정렬 방법

i개의 데이터( $a[0] \sim a[i-1]$ )가 이미 정렬되어 있다면

다음 데이터  $a[i]$ 를 오름차순을 유지하도록 삽입하는 방법

```
void insertion(int a[], int n)
{
    int i, j, idata;

    for (i=1; i <= n-1; i++) {
        idata = a[i];
        j = i-1;
        while (a[j] > idata && j >= 0) {
            a[j+1] = a[j]; j--;
        }
        a[j+1] = idata;
    }
}
```

n=7	0	1	2	3	4	5	6
a	17	20	30	40	25	10	50
i=4	17	20	25	30	40	10	50
i=5	10	17	20	25	30	40	50
i=6	10	17	20	25	30	40	50

# 삽입 정렬(Insertion Sort)

정렬할 데이터가 문자열인 경우 사전식 순서로 저장한다면

무엇을 변경해야 할까?

```
void insertion(int a[], int n)
{
    int i, j, idata;

    for (i=1; i <= n-1; i++) {
        idata = a[i];
        j = i-1;
        while (a[j] > idata && j >= 0) {
            a[j+1] = a[j]; j--;
        }
        a[j+1] = idata;
    }
}
```

```
void insertion_string(char a[][10], int n)
{
    int i, j;
    char idata[10];

    for (i=1; i <= n-1; i++) {
        strcpy(idata, a[i]);
        j = i-1;
        while ((strcmp(a[j], idata) > 0) && j >= 0) {
            strcpy(a[j+1], a[j]);
            j--;
        }
        strcpy(a[j+1], idata);
    }
}
```

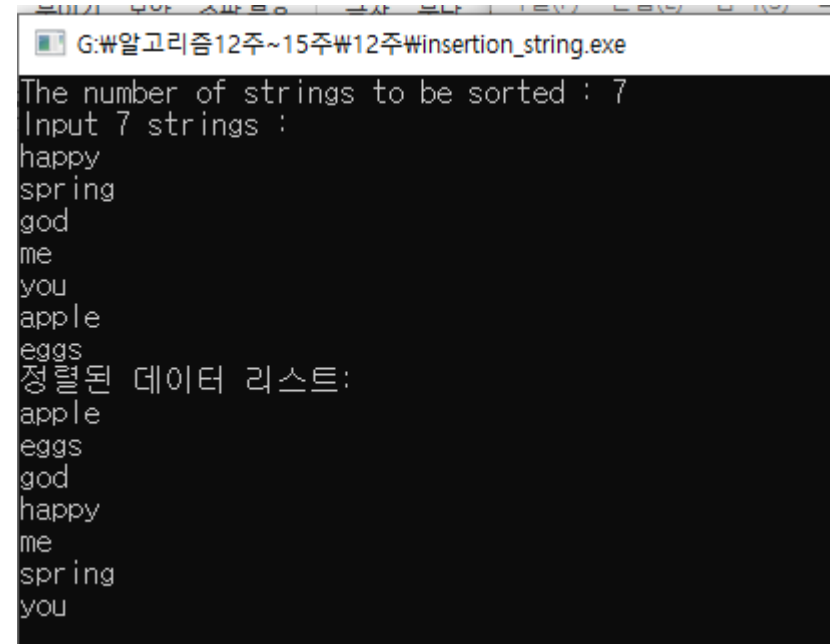
# 삽입 정렬(Insertion Sort)

정렬할 데이터가 문자열인 경우 사전식 순서로 저장한다면

무엇을 변경해야 할까?

```
void insertion_string(char a[][10], int n)
{
    int i, j;
    char idata[10];

    for (i=1; i <= n-1; i++) {
        strcpy(idata,a[i]);
        j = i-1;
        while ((strcmp(a[j],idata) > 0) && j >= 0) {
            strcpy(a[j+1], a[j]);
            j--;
        }
        strcpy(a[j+1],idata);
    }
}
```



```
G:\알고리즘12주~15주\12주\insertion_string.exe
The number of strings to be sorted : 7
Input 7 strings :
happy
spring
god
me
you
apple
eggs
정렬된 데이터 리스트:
apple
eggs
god
happy
me
spring
you
```

# 삽입정렬 정리 및 실습

1. 다음 함수 insertion()을 주어진 배열에 대하여 시뮬레이션해 보자.

$a[] = \{17, 20, 30, 40, 25, 10, 50\}$

```
void insertion(int a[], int n)
{
    int i, j, idata;

    for (i=1; i <= n-1; i++) {
        idata = a[i];
        j = i-1;
        while (a[j] > idata && j >= 0) {
            a[j+1] = a[j]; j--;
        }
        a[j+1] = idata;
    }
}
```

n=7	0	1	2	3	4	5	6
a	17	20	30	40	25	10	50
i=4							
i=5							
i=6							

# 삽입정렬 정리 및 실습

2. 10자 미만의 문자열을 사전식 오름차순으로 정렬하기 위한 다음 프로그램을 실행하고 기존의 int 데이터를 정렬한 다음 insertion 함수에서 변경해야하는 부분을 정리하시오

```
#define NUMSTRING 100

void insertion_string(char a[][10], int n);

main()
{
    char list[NUMSTRING][10];
    int i, n;

    printf("The number of strings to be sorted : ");
    scanf("%d", &n);
    for (i=0; i < n; i++)
        scanf("%s", list[i]);
    insertion_string(list, n);
    printf("정렬된 데이터 리스트: \n");
    for (i=0; i < n; i++)
        printf("%s\n", list[i]);
}
```

```
void insertion(int a[], int n)
{
    int i, j, idata;

    for (i=1; i <= n-1; i++) {
        idata = a[i];
        j = i-1;
        while (a[j] > idata && j >= 0) {
            a[j+1] = a[j]; j--;
        }
        a[j+1] = idata;
    }
}
```