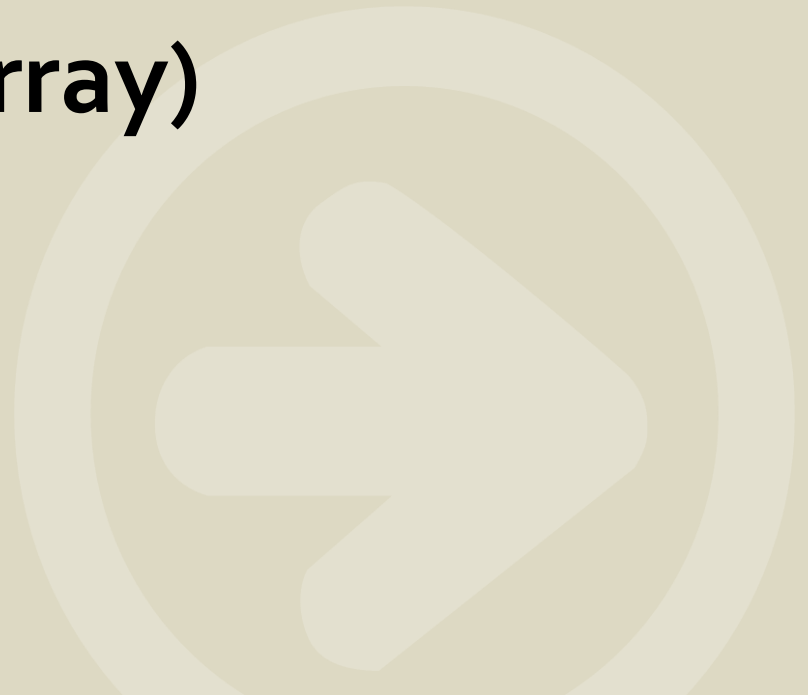


제 7 장

배 열(Array)





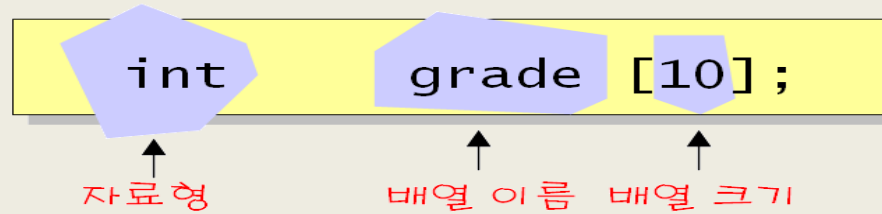
배열(Array)

- ❑ 2개 이상의 동일한 유형의 데이터가 순차적으로 저장될 수 있는 저장 공간을 나타내는 자료 구조
- ❑ 배열 내부의 각 데이터는 정수로 되어 있는 번호(인덱스, 색인)를 통해 액세스(읽기/쓰기) 한다
- ❑ 그러면 왜 배열이라는 데이터 유형을 사용할까?





배열의 선언



- ❑ 자료형: 배열을 구성하는 원소들이 int형라는 것을 의미
- ❑ 배열 이름: 배열을 칭하는 이름 (예, grade)으로 배열의 시작 주소이다
- ❑ 배열 크기: 배열을 이루는 원소의 개수
- ❑ 배열의 원소는 [] 안의 숫자로 지칭하며 이를 인덱스 (index)라고 하며, 인덱스는 항상 0부터 시작한다.

```
int score[60];           // 60개의 int형 값을 가지는 배열 grade
float cost[12];          // 12개의 float형 값을 가지는 배열 cost
char name[50];           // 50개의 char형 값을 가지는 배열 name
char src[10], dst[10];   // 2개의 문자형 배열을 동시에 선언
int index, days[7];      // 일반 변수와 배열을 동시에 선언
```



배열 원소의 액세스: 인덱스(index)

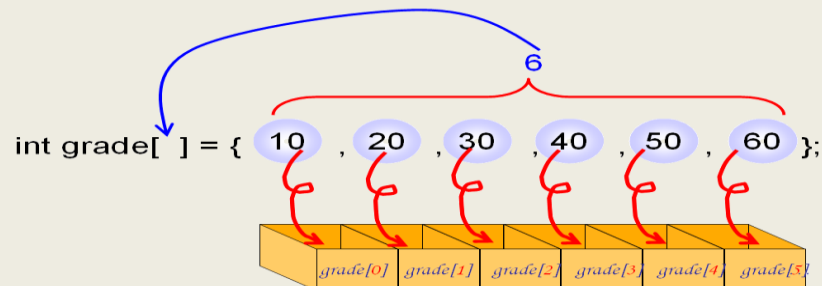
```
grade[5] = 80;  
grade[1] = grade[0];  
grade[i] = 100;           // i는 정수 변수  
grade[i+2] = 100;         // 수식이 인덱스가 된다.  
grade[index[3]] = 100;    // index[]는 정수 배열
```

배열의 초기화: 묵시적으로 0으로 초기화

- int grade[5] = { 10, 20, 30, 40, 50 };

- int grade[5] = { 10, 20, 30 };

배열의 크기가 주어지지 않으면 자동적으로 초기값의 개수가 배열의 크기로 잡힌다.



원소 참조 예제

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 10

int main(void)
{
    int grade[SIZE];
    int i;

    for(i = 0; i < SIZE; i++)
        grade[i] = rand() % 100;

    printf("=====₩n");
    printf("인덱스    값₩n");
    printf("=====₩n");

    for(i = 0; i < SIZE; i++)
        printf("%5d    %5d₩n", i, grade[i]);
    return 0;
}
```

인덱스	값
0	41
1	67
2	34
3	0
4	69
5	24
6	78
7	58
8	62
9	64

예 제

```
#include <stdio.h>
#define STUDENTS 5

int main(void)
{
    int grade[STUDENTS];
    int sum = 0;
    int i, average;

    for(i = 0; i < STUDENTS; i++)
    {
        printf("학생들의 성적을 입력하시오: ");
        scanf("%d", &grade[i]);
    }
    for(i = 0; i < STUDENTS; i++)
        sum += grade[i];

    average = sum / STUDENTS;
    printf("성적 평균= %d\n", average);
    return 0;
}
```

학생들의 성적을 입력하시오: 10
학생들의 성적을 입력하시오: 20
학생들의 성적을 입력하시오: 30
학생들의 성적을 입력하시오: 40
학생들의 성적을 입력하시오: 50
성적 평균 = 30



배열의 복사

```
int grade[SIZE];
```

```
int score[SIZE];
```

잘못된 방법

```
score = grade;
```

// 컴파일 오류!

```
#include <stdio.h>
```

```
#define SIZE 5
```

```
int main(void)
```

```
{
```

```
    int i;
```

```
    int a[SIZE] = {1, 2, 3, 4, 5};
```

```
    int b[SIZE];
```

```
    for(i = 0; i < SIZE; i++)
```

```
        b[i] = a[i];
```

올바른 방법

```
    return 0;
```

```
}
```



배열의 비교

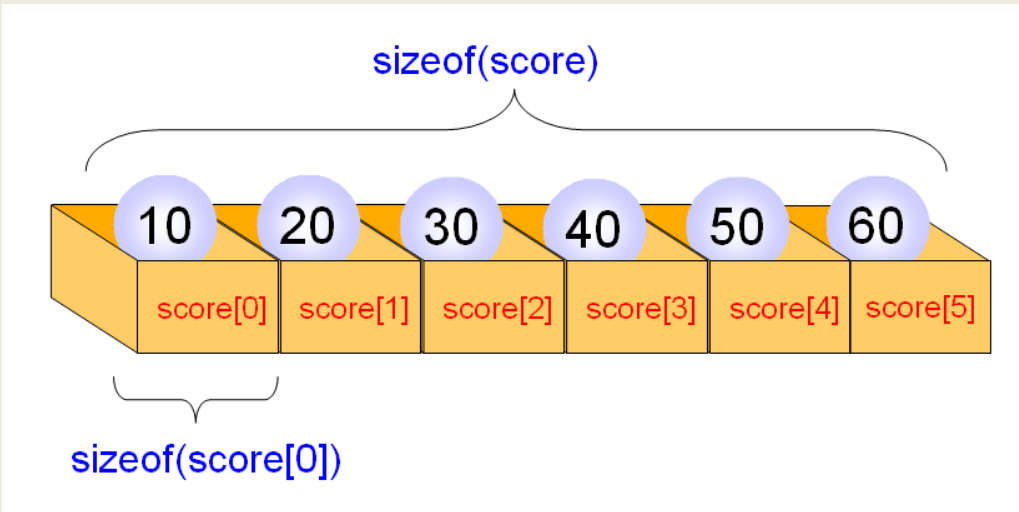
```
#include <stdio.h>
#define SIZE 5

int main(void)
{
    int i;
    int a[SIZE] = { 1, 2, 3, 4, 5 };
    int b[SIZE] = { 1, 2, 3, 4, 5 };

    if( a == b )           // ① 올바른지 않은 배열 비교
        printf("잘못된 결과입니다.\n");
    else
        printf("잘못된 결과입니다.\n");

    for(i = 0; i < SIZE ; i++) // ② 올바른 배열 비교
    {
        if ( a[i] != b[i] )
        {
            printf("a[]와 b[]는 같지 않습니다.\n");
            return 0;
        }
    }
    printf("a[]와 b[]는 같습니다.\n");
    return 0;
}
```


원소의 개수 계산



```
int score[] = { 10, 20, 30, 40, 50, 60 };
```

```
int i, size;
```

```
size = sizeof(score) / sizeof(score[0]);
```

배열 원소 개수 자동 계산

```
for(i = 0; i < size ; i++)
```

```
    printf("%d ", score[i]);
```



원소 수만큼 * 출력

```
#include <stdio.h>
#define STUDENTS 5

int main(void)
{
    int grade[STUDENTS] = { 30, 20, 10, 40, 50 };
    int i, s;

    for(i = 0; i < STUDENTS; i++)
    {
        printf("번호 %d: ", i);
        for(s = 0; s < grade[i]; s++)
            printf("*");
        printf("\n");
    }

    return 0;
}
```

```
번호 0: *****
번호 1: *****
번호 2: *****
번호 3: *****
번호 4: *****
```



최소값 탐색

```
#include <stdio.h>
#define SIZE 10

int main(void)
{
    int grade[SIZE];
    int i, min;

    for(i = 0; i < SIZE; i++)
    {
        printf("성적을 입력하시오: ");
        scanf("%d", &grade[i]);
    }

    min = grade[0];

    for(i = 1; i < SIZE; i++)
    {
        if( grade[i] < min )
            min = grade[i];
    }

    printf("최소값은 %d입니다.\n", min);
    return 0;
}
```

숫자를 입력하시오: 50
숫자를 입력하시오: 40
숫자를 입력하시오: 30
숫자를 입력하시오: 20
숫자를 입력하시오: 10
숫자를 입력하시오: 20
숫자를 입력하시오: 30
숫자를 입력하시오: 40
숫자를 입력하시오: 60
숫자를 입력하시오: 70
최소값은 10입니다.



주사위 면 빈도 계산

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 6

int main(void)
{
    int freq[SIZE] = { 0 };           // 주사위의 면의 빈도를 0으로 한다.
    int i;

    for(i = 0; i < 10000; i++)        // 주사위를 10000번 던진다.
        ++freq[ rand() % 6 ];        // 해당면의 빈도를 하나 증가한다.

    printf("=====\n");
    printf("면   빈도\n");
    printf("=====\n");

    for(i = 0; i < SIZE; i++)
        printf("%3d   %3d\n", i, freq[i]);

    return 0;
}
```

면	빈도
0	1657
1	1679
2	1656
3	1694
4	1652
5	1662



배열과 함수



- ❑ 함수의 매개변수로 배열 전체를 사용하는 경우
 - ❑ Call By Reference : 참조에 의한 호출
 - ❑ 함수 표현은 2가지를 사용할 수 있다 : 포인터, 배열 원본
- ❑ 함수의 인수로 배열 전체를 사용하는 경우
 - ❑ Call By Reference 방법을 사용하므로, 주소, 즉 배열의 이름을 인수로 사용하여야 한다
- ❑ 함수의 인수로 배열의 원소를 사용하는 경우
 - ❑ 하나의 데이터 이므로 다른 경우와 동일하게 Call By Value 방법을 사용

함수의 매개변수 설정 및 인수로 배열 전체를 사용

```
#include <stdio.h>
#define STUDENTS 5
int get_average(int score[], int n); // ①
```

```
int main(void)
{
    int grade[STUDENTS] = { 1, 2, 3, 4, 5 };
    int avg;

    avg = get_average(grade, STUDENTS);
    printf("평균은 %d입니다.\n", avg);
    return 0;
}
```

배열이 인수인 경우,
참조에 의한 호출

```
int get_average(int score[], int n) // ②
```

```
{
    int i;
    int sum = 0;

    for(i = 0; i < n; i++)
        sum += score[i];
    return sum / n;
}
```

배열의 복사본이 아닌
원본이 score[]로 전달

배열 전체와 배열 특정 원소를 인수로 사용하는 경우

```
#include <stdio.h>
#define SIZE 7

void square_array(int a[], int size);
void print_array(int a[], int size);
void square_element(int e);

int main(void)
{
    int list[SIZE] = { 1, 2, 3, 4, 5, 6, 7 };

    print_array(list, SIZE);
    square_array(list, SIZE); // 배열은 원본이 전달된다.
    print_array(list, SIZE);

    printf("%3d\n", list[6]);
    square_element(list[6]); // 배열 요소는 복사본이 전달된다.
    printf("%3d\n", list[6]);

    return 0;
}
```

```
void square_array(int a[], int size)
```

```
{  
    int i;  
  
    for(i = 0; i < size; i++)  
        a[i] = a[i] * a[i];  
}
```

```
void square_element(int e)
```

```
{  
    e = e * e;  
}
```

```
void print_array(int a[], int size)
```

```
{  
    int i;  
  
    for(i = 0; i < size; i++)  
        printf("%3d ", a[i]);  
    printf("\n");  
}
```

배열 원소의
사본이 *e*로 전달

배열의 원본을
사용

```
1 2 3 4 5 6 7  
1 4 9 16 25 36 49  
7  
7
```




정렬(sort)이란?

- 정렬은 데이터를 크기순으로 오름차순이나 내림차순으로 나열하는 것
- 정렬은 가장 기본적이고 중요한 알고리즘중의 하나
- 정렬은 자료 탐색에 있어서 필수적이다.

(예) 만약 사전에서 단어들이 정렬이 안되어 있다면?



비교	제조사	모델명	요약설명	판매가	업체수	출시
<input type="checkbox"/>	ROLLEI	D-41com	410만화소(0.56")/1.8"LCD/3배 줌/연사/CF카드	320,000	73	02년
<input type="checkbox"/>	카시오	QV-R40	413만화소(0.56")/1.6"LCD/3배 줌/동영상/히스토그램/앨범기능/SD, MMC카드	344,000	73	03년
<input type="checkbox"/>	파나소닉	DMC-LC43	423만화소(0.4")/1.5"LCD/3배 줌/동영상+녹음/연사/SD, MMC카드	348,000	36	03년
<input type="checkbox"/>	현대	DC-4311	400만화소(0.56")/1.6"LCD/3배 줌/동영상/SD, MMC카드	350,000	7	03년
<input type="checkbox"/>	삼성테크윈	Digimax420	410만화소(0.56")/1.5"LCD/3배 줌/동영상+녹음/음성메모/한글/SD카드	353,000	47	03년
<input type="checkbox"/>	니콘	Coolpix4300	413만화소(0.56")/1.5"LCD/3배 줌/동영상/연사/CF카드 Hot4	356,800	79	02년
<input type="checkbox"/>	올림푸스	뮤-20 Digital	423만화소(0.4")/1.5"LCD/3배 줌/동영상/연사/생활방수/xD카드	359,000	63	03년
<input type="checkbox"/>	코닥	LS-443(Dock포함)	420만화소/1.8"LCD/3배 줌/동영상+녹음/SD, MMC카드/Dock시스템	365,000	39	02년
<input type="checkbox"/>	올림푸스	C-450Z	423만화소(0.4")/1.8"LCD/3배 줌/동영상/연사/xD카드	366,000	98	03년
<input type="checkbox"/>	올림푸스	X-1	430만화소/1.5"LCD/3배 줌/동영상/연사/xD카드	367,000	19	03년
<input type="checkbox"/>	미놀타	DiMAGE-F100	413만화소(0.56")/1.5"LCD/3배 줌/동영상+녹음/음성메모/동체 추적AF/연사/SD, MMC카드	373,000	18	02년
<input type="checkbox"/>	삼성테크윈	Digimax410	410만화소(0.56")/1.6"LCD/3배 줌/동영상+녹음/음성메모/한글/CF카드	374,000	4	02년



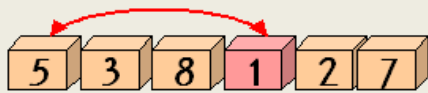
□ 정렬의 종류

- 선택형 정렬(Selection Sorts) : Selection sort, heap sort, ...
- 삽입형 정렬(Insertion Sorts) : Insertion sort, Shell sort, Tree sort, ...
- 교환형 정렬(Exchange Sorts) : Bubble sort, Quick sort, ...
- 병합형 정렬(Merge Sorts) : Merge sort

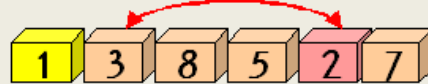


선택정렬(Selection sort)

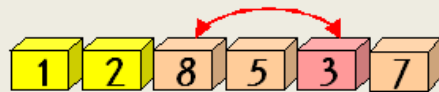
- 선택정렬(selection sort): 정렬이 안된 숫자들중에서 최소값(최대값)을 선택하여 배열의 첫 번째 요소와 교환 – Min(Max)-Selection sort



5과 1을 교환



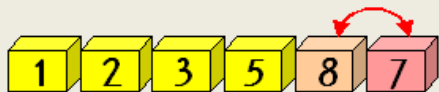
3과 2을 교환



8과 3을 교환



이미 제자리에 있음



8과 7을 교환



정렬 완료



Selection-Sort-Animation.gif

```
#include <stdio.h>
#define SIZE 10

void selection_sort(int list[], int n);
void print_list(int list[], int n);

int main(void)
{
    int grade[SIZE] = { 3, 2, 9, 7, 1, 4, 8, 0, 6, 5 };

    // 원래의 배열 출력
    printf("원래의 배열\n");
    print_list(grade, SIZE);

    selection_sort(grade, SIZE);

    // 정렬된 배열 출력
    printf("정렬된 배열\n");
    print_list(grade, SIZE);

    return 0;
}
```

```

void print_list(int list[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d ", list[i]);
    printf("\n");
}
void selection_sort(int list[], int n)           // 인덱스 기반으로 최소값 선택
{
    int i, j, temp, least;

    for(i = 0; i < n-1; i++)
    {
        least = i;

        for(j = i + 1; j < n; j++) // 최소값 탐색
            if(list[j] < list[least])
                least = j;
        // i번째 원소와 least 위치의 원소를 교환
        temp = list[i];
        list[i] = list[least];
        list[least] = temp;
    }
}

```

원래의 배열

3 2 9 7 1 4 8 0 6 5

정렬된 배열

0 1 2 3 4 5 6 7 8 9

```
void selection_sort1(int list[], int n) // 인덱스를 기반으로 최소값 선택
{
    int i, j, temp, min, min_index;

    for(i = 0; i < n-1; i++)
    {
        min_index = i;

        for(j = i + 1; j < n; j++) // 최소값 탐색
            if(list[j] < list[min_index]) {
                min_index = j ;
            }
        // i번째 원소와 least 위치의 원소를 교환
        temp = list[i];
        list[i] = list[min_index];
        list[min_index] = temp;
    }
}
```

```
void selection_sort2(int list[], int n) // 데이터 자체를 기반으로 최소값 선택
{
    int i, j, temp;

    for(i = 0; i < n; i++)
    {
        for(j = i + 1; j < n; j++) // 최소값 탐색
            if(list[j] < list[i]) {
                temp = list[i];
                list[i] = list[j];
                list[j] = temp;
            }
    }
}
```

◆ selection_sort1과 selection_sort2의 차이는 무엇일까?

삽입정렬(Insertion Sort)

Algorithm (Sorted portion/Unsorted portion)

```
InsertionSort(int a[], int n) {  
    int wall, j, pivot;
```

```
    for(wall=1; i<n; wall++) {  
        pivot = a[wall] ;    // wall 미만의 부분 = sorted portion  
        j = wall-1 ;  
        while (j >= 0 && (pivot < a[j])) {  
            a[j+1] = a[j];  
            j -- ;  
        }  
        a[j+1] = pivot ;
```


sorted portion | unsorted portion

pivot = a[wall]

6 5 3 1 8 7 2 4

<삽입 정렬>

j = wall-1

a[j+1] = a[j] (pivot < a[j] 이므로)

5 6 3 1 8 7 2 4

<삽입 정렬>

a[j+1] = pivot

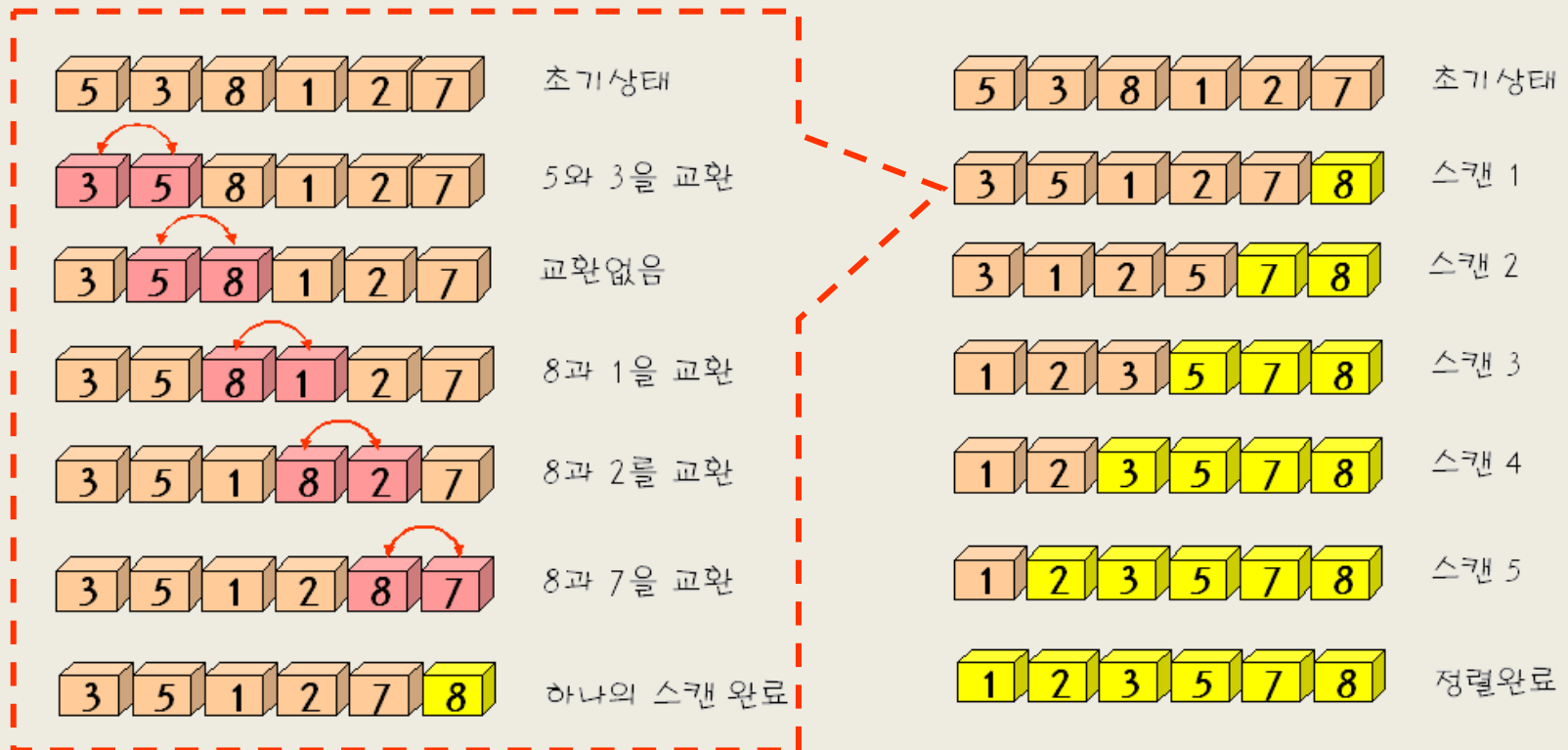
3 5 6 1 8 7 2 4

<삽입 정렬>



버블정렬(bubble sort)

- ❑ 인접한 레코드가 순서대로 되어 있지 않으면 교환
- ❑ 전체가 정렬될 때까지 비교/교환 계속



버블 정렬 프로그램

```
void BubbleSort(int a[], int n)
{
    int i, scan, temp;

    // 스캔 회수를 제어하기 위한 루프
    for(scan = 0; scan < n-1; scan++)
    {
        // 인접값 비교 회수를 제어하기 위한 루프
        for(i = 0; i < n-1; i++)
        {
            // 인접값 비교 및 교환
            if( a[i] > a[i+1] )
            {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }
        }
    }
}
```



□ 좀 더 효율적 알고리즘

```
void BubbleSort(int a[], int n)
{
    int i, scan, temp;

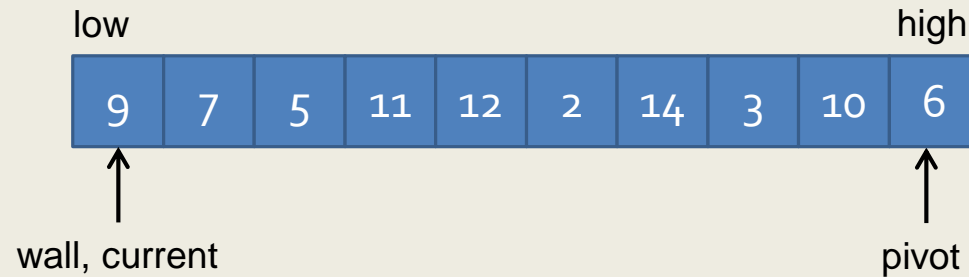
    // 스캔 회수를 제어하기 위한 루프
    for(scan = 0; scan < n-1; scan++)
    {
        // 인접값 비교 회수를 제어하기 위한 루프
        for(i = 0; i < n-1-scan ; i++)
        {
            // 인접값 비교 및 교환
            if( a[i] > a[i+1] )
            {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }
        }
    }
}
```



Quick Sort

- ❑ 퀵 정렬은 연속적인 분할-정복(divide-and-conquer) 알고리즘을 통해 이루어진다.
- ❑ 축(Pivot)값을 중심으로 왼쪽은 이 축 값보다 작은 값으로 오른쪽은 모두 이 축 값보다 큰 값을 배열시키는 것이다.
- ❑ 축 값의 왼쪽과 오른쪽 부분에 대해 또다시 분할 과정을 적용하여 분할의 크기가 1이 될 때까지 반복하면 전체적 정렬이 완료.
- ❑ 재귀(recursive) 알고리즘 사용
 - 배열 a를 pivot을 기준으로 2 부분으로 분할하여 최종적인 pivot의 위치를 결정한다 (예를 들면 mid)
 - 왼쪽파트에 대해 동일한 퀵 정렬 알고리즘 적용
 - 오른쪽 파트에 대한 동일한 퀵 정렬 알고리즘 적용

Divide Algorithm (with video)-Using Lomuto Partition Scheme

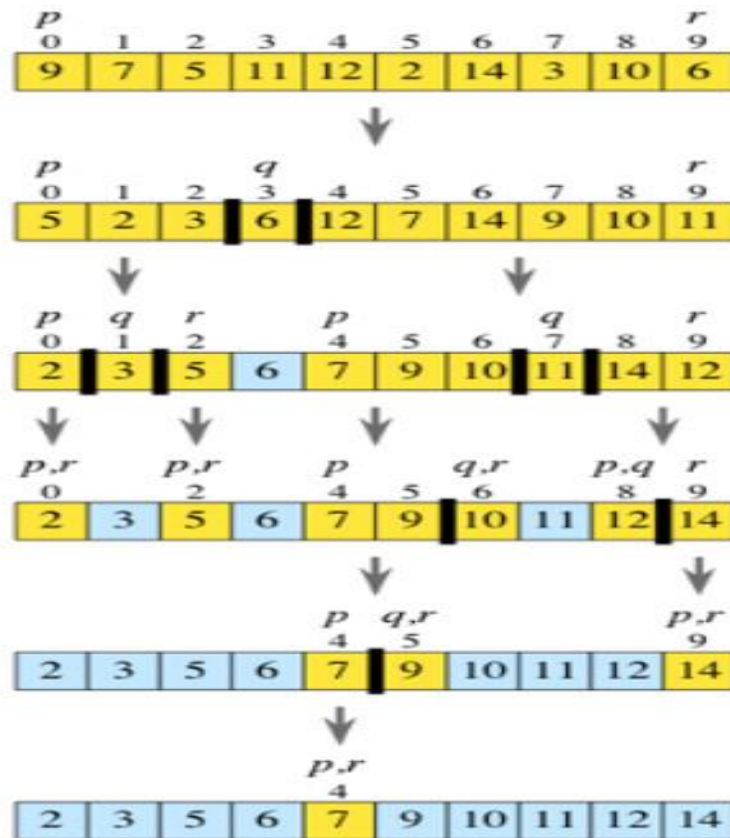


```
int Divide(int array[], int low, int high) {  
    int wall, current, pivot;  
    pivot = array[high];  
    wall = low ; //place for swapping  
    for (current = low; current < high; current++)  
        if(array[current] <= pivot) {  
            swap(&array[wall], &array[current]);  
            wall++;  
        }  
    swap(&array[wall], &array[high]);  
    return wall ;  
}
```

Quick Sort Using Lomuto Partition Scheme

```
void QuickSort(int array[], int low, int high) {
    int p;
    if(low < high) {
        p = divide(array, low, high);
        QuickSort(array, low, p - 1);
        QuickSort(array, p + 1, high);
    }
}

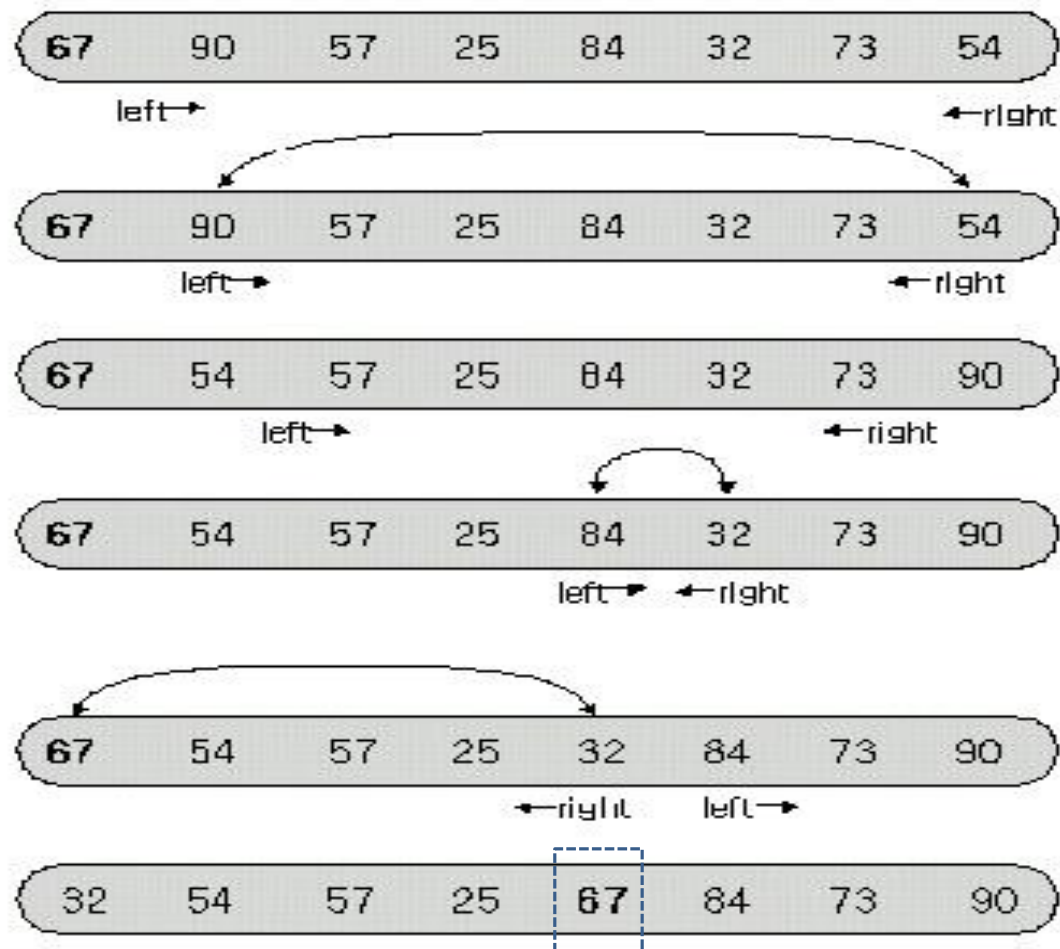
int divide(int array[], int low, int high) {
    int wall, current, pivot;
    pivot = array[high] ;
    wall = low ;                //place for swapping
    for (current = low; current < high; current++)
        if(array[current] <= pivot ) {
            swap(&array[wall], &array[current]);
            wall++;
        }
    swap(&array[wall], &array[high]);
    return wall ;
}
```



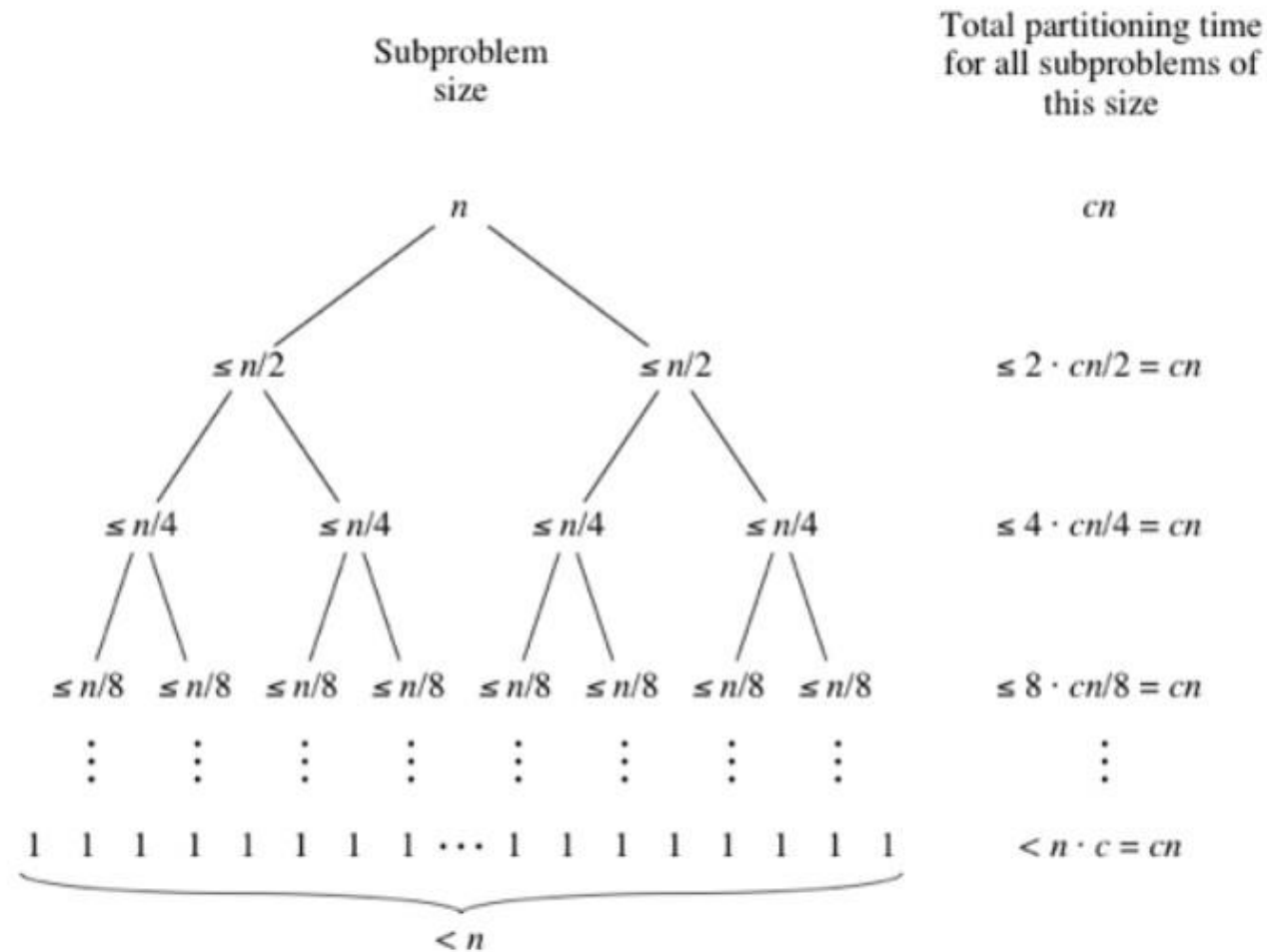
Quick Sort Using Hoare Partition Scheme

```
void QuickSort(int array[], int low, int high) {
    int p;
    if(low < high) {
        p = divide(array, low, high);
        QuickSort(array, low, p);
        QuickSort(array, p + 1, high);
    }
}

int divide(int array[], int low, int high) {
    int pivot, left, right;
    pivot = array[low] ;
    left = low - 1 ;
    right = high + 1 ;
    while(1) {
        do right-- ;
        while(array[right] > pivot)
        do left++ ;
        while(array[left] < pivot)
        if(left < right)
            swap(&array[left], &array[right]);
        else
            return right;
    }
}
```



Best-case running time : $O(n \log n)$



C 언어가 제공하는 qsort()

```
#include <stdio.h>
#include <search.h>
#include <string.h>

void main(void) {

    // qsort 라이브러리를 사용
    int array[] = { 3, 5, 6, 3, 1, 2, 7, 6, 7, 4, 8, 9, 3 };
    qsort(array, 13, sizeof(int), strcmp);

    for (i = 0; i < 13; i++){
        printf("%d\t", array[i]);
    }

    return 0;
}
```



Merge Sort(병합정렬)

```
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

```
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}
```

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}
```

```

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    /* create temp arrays */
    int L[n1], R[n2];
    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
}

```

```

/* Copy the remaining elements of L[], if there are any */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy the remaining elements of R[], if there are any */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}

```

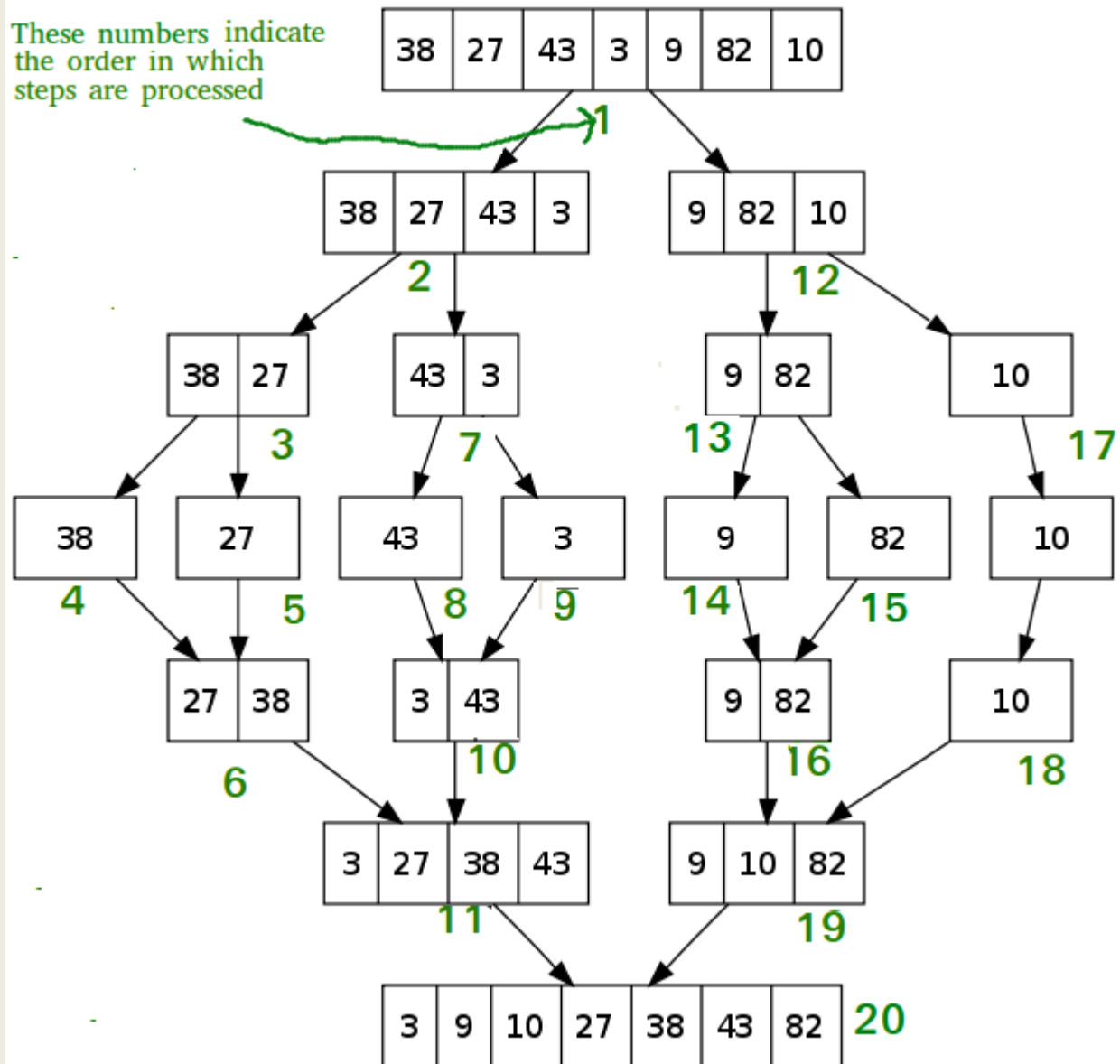
Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13

These numbers indicate
the order in which
steps are processed



순차 탐색(Sequential Search)

```
#include <stdio.h>
#define SIZE 6

int seq_search(int list[], int n, int key);

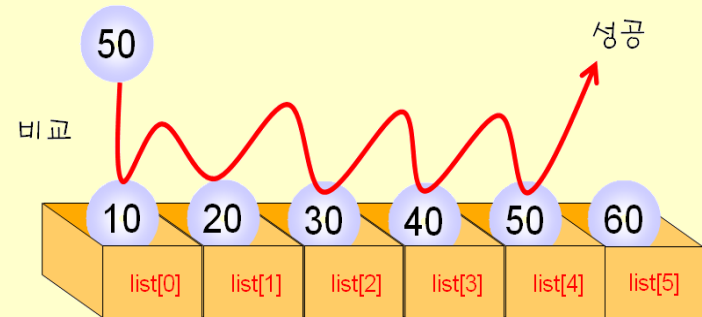
int main(void)
{
    int key;
    int grade[SIZE] = { 10, 20, 30, 40, 50, 60 };

    printf("탐색할 값을 입력하시오:");
    scanf("%d", &key);
    printf("탐색 결과 = %d\\n", seq_search(grade, SIZE, key));

    return 0;
}

int seq_search(int list[], int n, int key)
{
    int i;

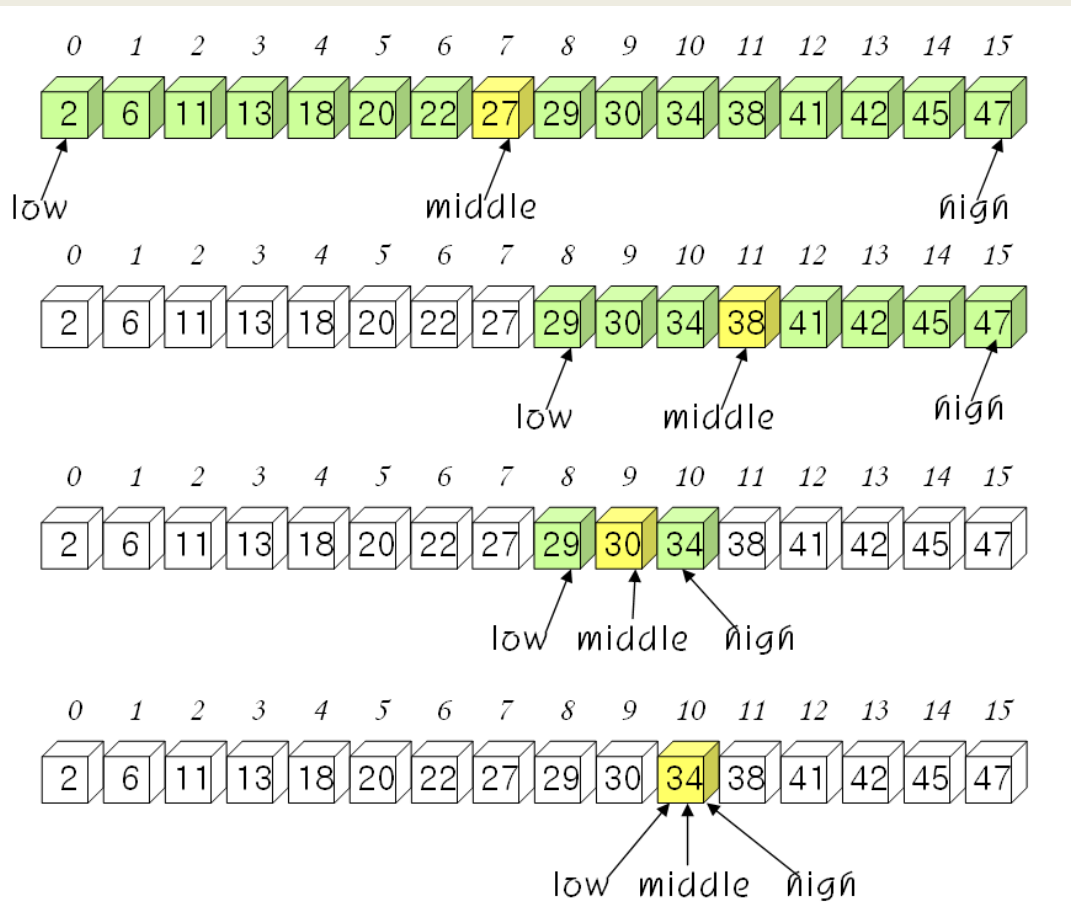
    for(i = 0; i < SIZE; i++)
        if(list[i] == key)
            return i; // 탐색이 성공하면 인덱스 반환
    return -1;        // 탐색이 실패하면 -1 반환
}
```





이진 탐색(Binary Search)

- 이진 탐색(binary search): 정렬된 배열의 중앙에 위치한 원소와 비교 되
풀이-그러므로 **먼저 정렬이 되어 있어야 한다**





❑ 이진 탐색 알고리즘(binary search algorithm)

- ❑ 정렬된 리스트에서 특정한 값의 위치를 찾는 알고리즘
- ❑ 정렬된 리스트에만 사용할 수 있다는 단점이 있지만, 검색이 반복될 때마다 목표 값을 찾을 확률은 두 배가 되므로 속도가 빠르다는 장점이 있다
- ❑ 분할 정복 알고리즘

반복문 사용

```
int binary_search(int list[], int n, int key)
{
    int low, high, middle;

    low = 0;
    high = n-1;

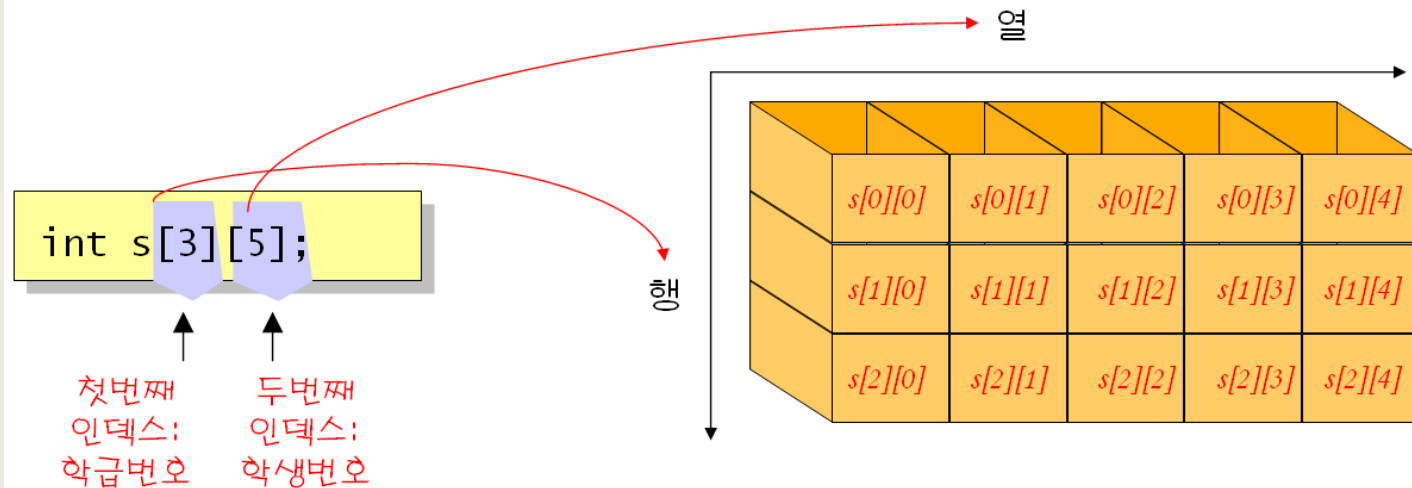
    while( low <= high ){        // 아직 숫자들이 남아있으면
        middle = (low + high)/2;  // 중간 요소 결정
        if( key == list[middle] ) // 일치하면 탐색 성공
            return middle;
        else if( key > list[middle] ) // 중간 원소보다 크다면
            low = middle + 1;      // 새로운 값으로 low 설정
        else
            high = middle - 1;    // 새로운 값으로 high 설정
    }
    return -1;
}
```

재귀함수 이용

```
BinarySearch(list[], key, low, high) {  
    if (high < low)  
        return -1 ;                // not found  
    mid = (low + high) / 2 ;  
  
    if (list[mid] > key)  
        return BinarySearch(list, key, low, mid-1) ;  
  
    else if (list[mid] < key)  
        return BinarySearch(list, key, mid+1, high) ;  
  
    else  
        return mid ;                // found  
}
```

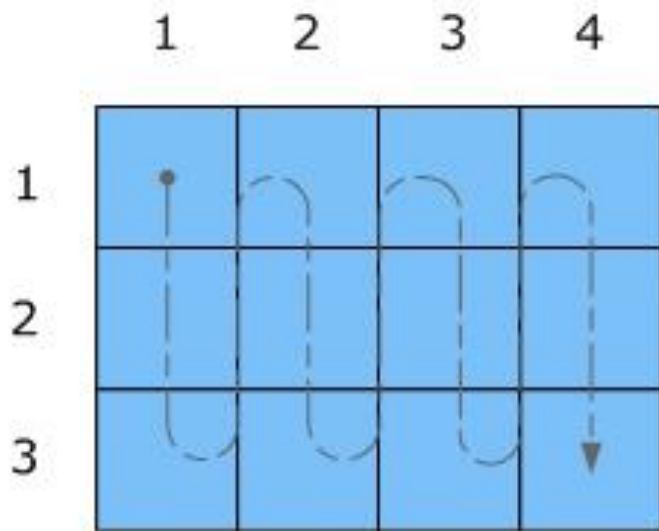
2차원 배열

```
int s[10];      // 1차원 배열  
int s[3][10];   // 2차원 배열  
int s[5][3][10]; // 3차원 배열
```

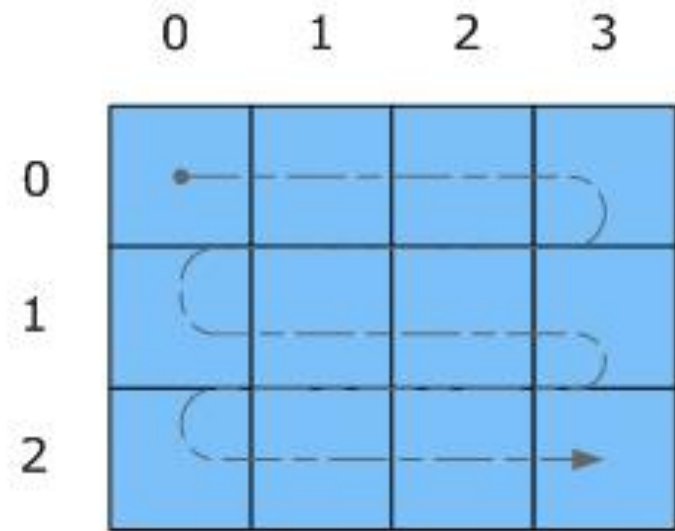




- ❑ 2차원 배열은 1차원적으로 구현된다
- ❑ Row-major vs. Column-major



A: Column-major order (Fortran-style)



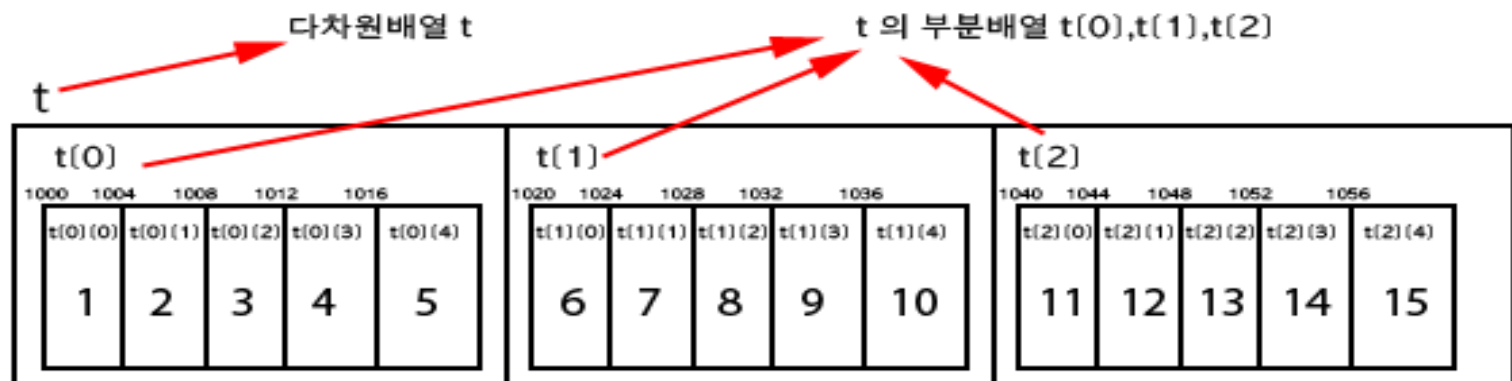
B: Row-major order (C-style)

```
int t(3)(5)={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
```

한 원소의 크기

int t (3) (5);

한원소의 크기(int형 5개)



Two Dimensional Array

1	5	3	6
3	2	38	64
22	76	82	99
0	106	345	54

User's view (abstraction)

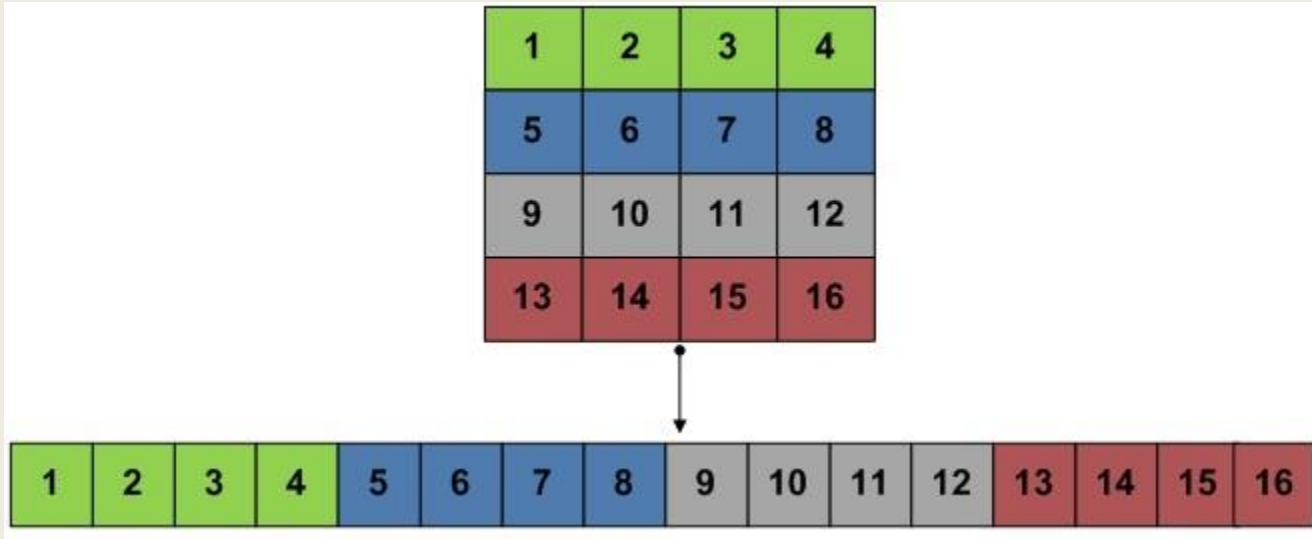
1	5	3	6	3	2	38	64	22	76	82	99	0	106	345	54
---	---	---	---	---	---	----	----	----	----	----	----	---	-----	-----	----

System's view
(implementation)

Offset of $a[i][j]$?

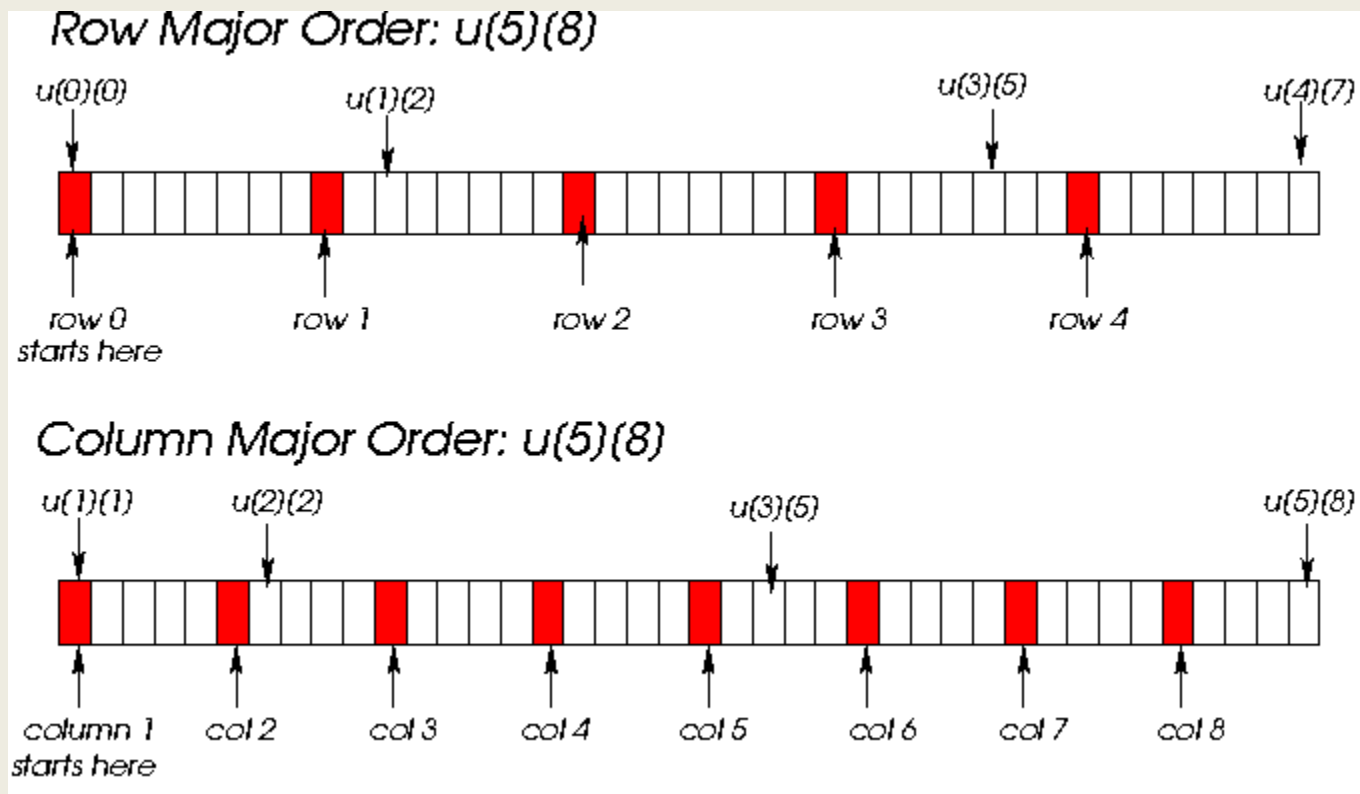


Example 1





□ Example 2



a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

	주소의 배열식 표기법	주소의 배열 & 포인터식 표기법	주소의 포인터식 표기법	메모리	값의 배열식 표기법	값의 배열 & 포인터식 표기법	값의 포인터식 표기법
a	&a[0][0]	a[0]	*(a+0)+0	1000	a[0][0]	*(a[0])	*(*(a+0)+0)
	&a[0][1]	a[0]+1	*(a+0)+1	1004	a[0][1]	*(a[0]+1)	*(*(a+0)+1)
	&a[0][2]	a[0]+2	*(a+0)+2	1008	a[0][2]	*(a[0]+2)	*(*(a+0)+2)
	&a[0][3]	a[0]+3	*(a+0)+3	1012	a[0][3]	*(a[0]+3)	*(*(a+0)+3)
a+1	&a[1][0]	a[1]	*(a+1)+0	1016	a[1][0]	*(a[1])	*(*(a+1)+0)
	&a[1][1]	a[1]+1	*(a+1)+1	1020	a[1][1]	*(a[1]+1)	*(*(a+1)+1)
	&a[1][2]	a[1]+2	*(a+1)+2	1024	a[1][2]	*(a[1]+2)	*(*(a+1)+2)
	&a[1][3]	a[1]+3	*(a+1)+3	1028	a[1][3]	*(a[1]+3)	*(*(a+1)+3)
a+2	&a[2][0]	a[2]	*(a+2)+0	1032	a[2][0]	*(a[2])	*(*(a+2)+0)
	&a[2][1]	a[2]+1	*(a+2)+1	1036	a[2][1]	*(a[2]+1)	*(*(a+2)+1)
	&a[2][2]	a[2]+2	*(a+2)+2	1040	a[2][2]	*(a[2]+2)	*(*(a+2)+2)
	&a[2][3]	a[2]+3	*(a+2)+3	1044	a[2][3]	*(a[2]+3)	*(*(a+2)+3)

```

C:\WINDOWS\system32\cmd.exe

&array      : 0012FF40      &array+1    : 0012FF70
array       : 0012FF40      array+1     : 0012FF50
array[0]    : 0012FF40      array[0]+1 : 0012FF44

```

2차원 배열의 활용

```
#include <stdio.h>

int main(void)
{
    int s[3][5];    // 2차원 배열 선언
    int i, j;        // 2개의 인덱스 변수
    int value = 0;    // 배열 원소에 저장되는 값

    for(i=0;i<3;i++)
        for(j=0;j<5;j++)
            s[i][j] = value++;

    for(i=0;i<3;i++)
        for(j=0;j<5;j++)
            printf("%d\\n", s[i][j]);

    return 0;
}
```

2차원 배열의 초기화

```
int s[3][5] = {  
    { 0, 1, 2, 3, 4 }, // 첫 번째 행의 원소들의 초기값  
    { 10, 11, 12, 13, 14 }, // 두 번째 행의 원소들의 초기값  
    { 20, 21, 22, 23, 24 } // 세 번째 행의 원소들의 초기값  
};
```

```
int s[ ][5] = {  
    { 0, 1, 2, 3, 4 }, // 첫 번째 행의 원소들의 초기값  
    { 10, 11, 12, 13, 14 }, // 두 번째 행의 원소들의 초기값  
    { 20, 21, 22, 23, 24 }, // 세 번째 행의 원소들의 초기값  
};
```

```
int s[ ][5] = {  
    { 0, 1, 2 },           // 첫 번째 행의 원소들의 초기값  
    { 10, 11, 12 },       // 두 번째 행의 원소들의 초기값  
    { 20, 21, 22 }        // 세 번째 행의 원소들의 초기값  
};
```

```
int s[ ][5] = {  
    0, 1, 2, 3, 4,         // 첫 번째 행의 원소들의 초기값  
    5, 6, 7, 8, 9,        // 두 번째 행의 원소들의 초기값  
};
```

3차원 배열

```
int s [6][3][5];
```

첫번째 두번째 세번째
인덱스: 인덱스: 인덱스:
학년번호 학급번호 학생번호

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    int s[3][3][3];    // 3차원 배열 선언  
    int x, y, z;        // 3개의 인덱스 변수  
    int i = 1;          // 배열 원소에 저장되는 값
```

```
    for(z=0;z<3;z++)  
        for(y=0;y<3;y++)  
            for(x=0;x<3;x++)  
                s[z][y][x] = i++;
```

```
    return 0;
```

```
}
```

다차원 배열 인수

```
#include <stdio.h>
#define YEARS    3
#define PRODUCTS 5

int sum(int grade[][PRODUCTS]);

int main(void)
{
    int sales[YEARS][PRODUCTS] = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
    int total_sale;
    total_sale = sum(sales);
    printf("총매출은 %d입니다.\n", total_sale);
    return 0;
}

int sum(int grade[][PRODUCTS])
{
    int y, p;
    int total = 0;
    for(y = 0; y < YEARS; y++)
        for(p = 0; p < PRODUCTS; p++)
            total += grade[y][p];
    return total;
}
```

첫번째 인덱스의 크기는
적지 않아도 된다.

다차원 배열 예제

학급 0의 평균 성적 = 2
학급 1의 평균 성적 = 12
학급 2의 평균 성적 = 22
전체 학생들의 평균 성적 = 12


```
#include <stdio.h>
#define CLASSES 3
#define STUDENTS 5

int main(void)
{
    int s[CLASSES][STUDENTS] = {
        { 0, 1, 2, 3, 4 },    // 첫번째 행의 원소들의 초기값
        { 10, 11, 12, 13, 14 }, // 두번째 행의 원소들의 초기값
        { 20, 21, 22, 23, 24 }, // 세번째 행의 원소들의 초기값
    };
    int clas, student, total, subtotal;
    total = 0;
    for(clas = 0; clas < CLASSES; clas++)
    {
        subtotal = 0;
        for(student = 0; student < STUDENTS; student++)
            subtotal += s[clas][student];
        printf("학급 %d의 평균 성적 = %d\n", clas, subtotal / STUDENTS);
        total += subtotal;
    }
    printf("전체 학생들의 평균 성적 = %d\n", total/(CLASSES * STUDENTS));
    return 0;
}
```


다차원 배열을 이용한 행렬의 표현

```
#include <stdio.h>
#define ROWS 3
#define COLS 3

int main(void)
{
    int A[ROWS][COLS] = { { 2,3,0 }, { 8,9,1 }, { 7,0,5 } };
    int B[ROWS][COLS] = { { 1,0,0 }, { 1,0,0 }, { 1,0,0 } };
    int C[ROWS][COLS];
    int r,c;
    // 두개의 행렬을 더한다.
    for(r = 0; r < ROWS; r++)
        for(c = 0; c < COLS; c++)
            C[r][c] = A[r][c] + B[r][c];
    // 행렬을 출력한다.
    for(r = 0; r < ROWS; r++)
    {
        for(c = 0; c < COLS; c++)
            printf("%d ", C[r][c]);
        printf("\n");
    }
    return 0;
}
```



```
3 3 0
9 9 1
8 0 5
```