

## C - 알고리즘 2



## 정렬, 탐색 알고리즘



# 정렬 알고리즘

- 정렬(sorting)

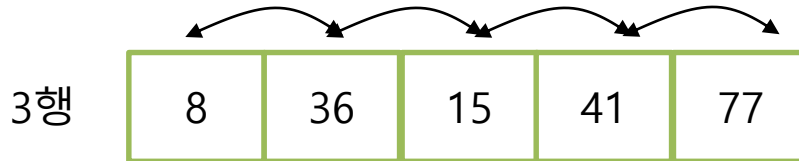
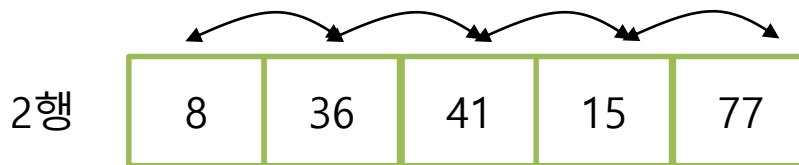
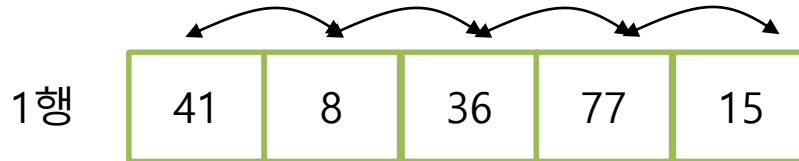
정렬(sorting)은 이름, 학번, 키 등 핵심 항목(key)의 대소 관계에 따라 데이터 집합을 일정한 순서로 줄지어 늘어서도록 바꾸는 작업을 말한다.  
이 알고리즘을 이용해 데이터를 정렬하면 검색을 더 쉽게 할 수 있다.

키값이 작은 값을 앞쪽에 놓으면 오름차순(ascending order) 정렬, 그 반대로 놓으면 내림차순(descending order) 정렬이라고 부른다.



# 정렬 알고리즘

- 버블 정렬(bubble sorting)
  - 리스트에서 인접한 두 개의 요소를 비교하여 자리를 바꾸는 방식
  - 요소의 개수가  $n$ 개인 배열에서  $n-1$ 회 비교 교환함



4행 교환 없음

5행 교환 없음

## 정렬 결과

[8, 36, 41, 15, 77]

[8, 36, 15, 41, 77]

**[8, 15, 36, 41, 77] – 완료!**



# 정렬 알고리즘

- 버블 정렬(bubble sorting)

```
//버블 정렬 - 오름차순
int arr[5] = {41, 8, 36, 77, 15};
int i, j, temp;

//비교와 교환 반복
for (i = 0; i < 5; i++) {
    for (j = 0; j < 4; j++) {
        if (arr[j] > arr[j+1]) {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
        }
    }
}
```



# 정렬 알고리즘

- 버블 정렬(bubble sorting)

```
/*  
    i=0, j=0, 41>8, 8,41,36,77,15  
        j=1, 41>36, 8,36,41,77,15  
        j=2, 41>77, 교환없음  
        j=3, 77>15, 8,36,41,15,77  
    i=1, j=0, 8>36, 교환없음  
        j=1, 36>41, 교환없음  
        j=2, 41>15, 8,36,15,41,77  
        j=3, 41>77, 교환없음  
    i=2, j=0, 8>36, 교환없음  
        j=1, 36>15, 8,15,36,41,77  
    i=3, 교환없음  
    i=4, 교환없음  
*/
```

```
//출력
```

```
for (i = 0; i < 5; i++) {  
    printf("%d ", arr[i]);  
}
```

8 15 36 41 77



# 정렬 알고리즘

- 버블 정렬(bubble sorting) – 함수로 정의

```
void bubbleSorting(int a[], int n) {  
    int i, j, temp;  
  
    for (i = 0; i < n; i++) {  
        for (j = 0; j < n - 1; j++) {  
            if (a[j] > a[j + 1]) {  
                temp = a[j];  
                a[j] = a[j + 1];  
                a[j + 1] = temp;  
            }  
        }  
    }  
}
```



# 정렬 알고리즘

- 버블 정렬(bubble sorting) – 함수로 정의

```
int arr[5] = {41, 8, 36, 77, 15};
int size, i;

size = sizeof(arr) / sizeof(arr[0]);

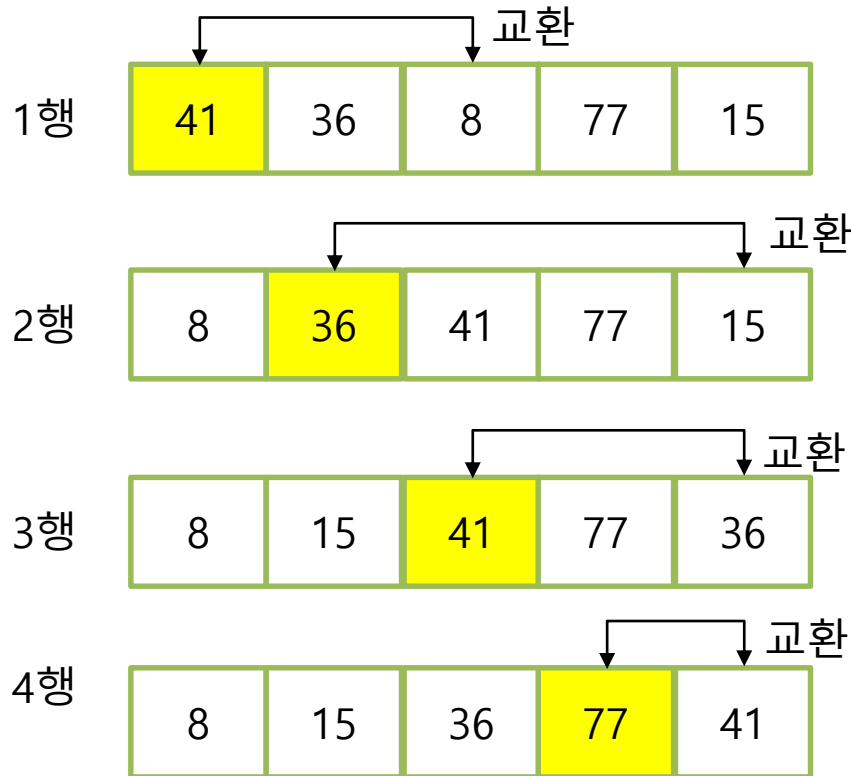
//버블 정렬 함수 호출
bubbleSorting(arr, size);

for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
}
```



# 정렬 알고리즘

- 선택 정렬(selection sorting)
  - 리스트에서 가장 작은 값을 찾아 맨 앞으로 보내는 방식



$i = 0$  (첫번째 위치 고정)  
최소값 8 찾음. ( $\text{min\_idx}=2$ )  
**[8, 36, 41, 77, 15]**

$i = 1$  (첫번째 위치 고정)  
최소값 15 찾음. ( $\text{min\_idx}=4$ )  
**[8, 15, 41, 77, 36]**

$i = 2$  (첫번째 위치 고정)  
최소값 36 찾음. ( $\text{min\_idx}=4$ )  
**[8, 15, 36, 77, 41]**

$i = 3$  (첫번째 위치 고정)  
최소값 41 찾음. ( $\text{min\_idx}=4$ )  
**[8, 15, 36, 41, 77] - 완료**



# 정렬 알고리즘

- 선택 정렬(selection sorting)

```
int arr[5] = { 41, 36, 8, 77, 15 };
int i, j, temp;

//비교와 교환 반복
for (i = 0; i < 4; i++) {
    int minIdx = i; //현재 위치(행)를 최소값으로 설정
    for (j = i + 1; j < 5; j++) {
        if (arr[j] < arr[minIdx])
            minIdx = j; //비교후 최소값 위치 변경
    }

    //교환 처리
    temp = arr[i];
    arr[i] = arr[minIdx];
    arr[minIdx] = temp;
}
```



# 정렬 알고리즘

- 선택 정렬(selection sorting)

```
//정렬 후 출력  
for (i = 0; i < 5; i++)  
    printf("%d ", arr[i]);
```

```
8 15 36 41 77
```



# 정렬 알고리즘

- 선택 정렬(selection sorting) – 함수로 구현

```
void selectionSorting(int a[], int n) {  
    int i, j, temp;  
  
    //비교와 교환 반복  
    for (i = 0; i < n - 1; i++) {  
        int minIdx = i; //현재 위치(행)를 최소값으로 설정  
        for (j = i + 1; j < n; j++) {  
            if (a[j] < a[minIdx])  
                minIdx = j; //비교후 최소값 위치 변경  
        }  
  
        temp = a[i];  
        a[i] = a[minIdx];  
        a[minIdx] = temp;  
    }  
}
```



# 정렬 알고리즘

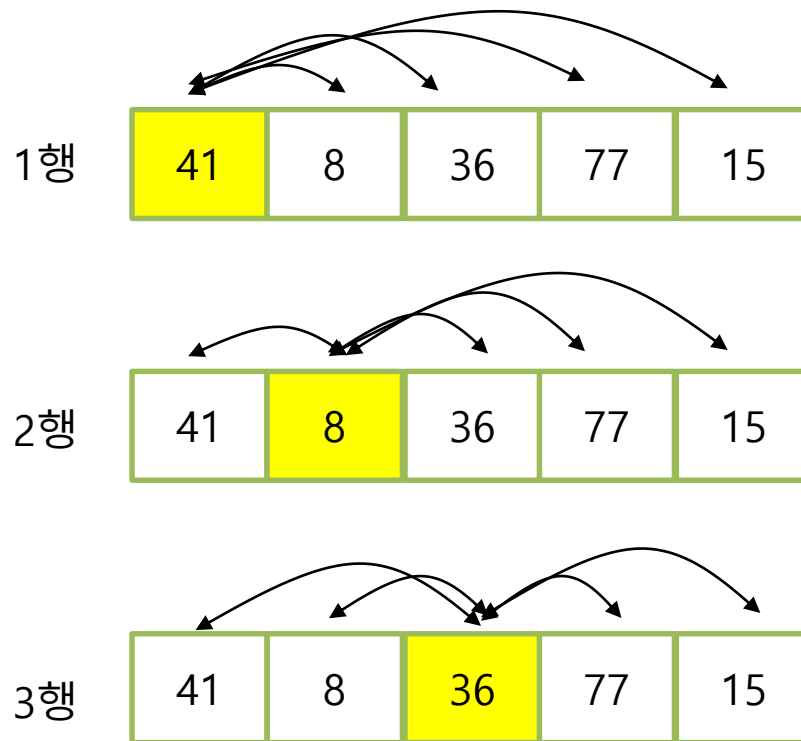
- 선택 정렬(selection sorting) – 함수로 구현

```
int arr[5] = { 41, 36, 8, 77, 15 };  
int i, j, temp, size;  
  
size = sizeof(arr) / sizeof(arr[0]);  
  
//선택 정렬 함수 호출  
selectionSorting(arr, size);  
  
//정렬 후 출력  
for (i = 0; i < size; i++)  
    printf("%d ", arr[i]);
```



# 정렬 알고리즘

- 순위 정하기



## 비교 결과

[2, 1, 1, 1, 1]

[2, 5, 1, 1, 1]

[2, 5, 3, 1, 1]



# 정렬 알고리즘

- 순위 정하기

```
int arr[] = { 41, 8, 36, 77, 15 };
int rank[] = { 1, 1, 1, 1, 1 };
int size = sizeof(arr) / sizeof(arr[0]);
int i, j;

//비교후 순위 결정
for (i = 0; i < size; i++){
    for (j = 0; j < size; j++) {
        if (arr[i] < arr[j])
            rank[i] = rank[i] + 1;
    }
}

//순위 출력
for (i = 0; i < size; i++) {
    printf("%d ", rank[i]); //2 5 3 1 4
}
```



# 알고리즘 기초

- 배열에서 값 찾기 1

```
int a[] = { 9, 8, 7, 6, 7 };
int i;
int count = 0;

//7이 몇 개인지 세기
for (i = 0; i < 5; i++){
    if (a[i] == 7) {
        printf("7 발견!\n");
        count++;
    }
}
printf("7을 %d개 발견!", count);
```



# 알고리즘 기초

- 배열에서 값 찾기 2

```
//7을 하나 발견하면 종료
int sw = 0; //상태(토글) 변수
for (i = 0; i < 7; i++) {
    if (a[i] == 7) {
        printf("7 발견!\n");
        sw = 1;
        break;
    }
}

if(sw == 0)
    printf("7을 발견 못함!\n");
```





# 알고리즘 기초

- 문자열 역순으로 읽기

```
char a1[] = "DOG";  
char a2[10]; //충분한 크기 확보  
int i;  
  
//a1을 a2에 거꾸로 복사  
for (i = 0; i < 4; i++) {  
    a2[i] = a1[2 - i];  
}  
a2[3] = '\0'; //문자열 끝에 널문자 추가  
  
printf("%s를 거꾸로 읽으면 %s\n", a1, a2);
```

DOG를 거꾸로 읽으면 GOD



# 알고리즘 기초

- 문자열 역순으로 읽기

```
int n;  
char a[] = "DOG";  
char b[10];  
  
n = strlen(a); //3 - a의 개수  
  
for (i = n-1; i >= 0; i--) {  
    b[n-1-i] = a[i];  
}  
b[n] = '\0';  
  
printf("%s를 거꾸로 읽으면 %s\n", a, b);
```



# 알고리즘 기초

- 두 문자열 연결하기

```
/*  
- 두 개의 문자열을 연결하여 하나의 문자열로 만들기  
a 배열은 충분히 커야 합니다.  
"smart"(5자) + "phone"(5자) + '\0' = 총 11바이트가 필요  
*/
```

```
int i = 0, j = 0;  
char a[12] = "smart";  
char b[] = "phone";  
  
printf("%s+%s=", a, b);  
while (a[i] != '\0')  
    i++;  
//printf("\ni=%d\n", i); //5
```



# 알고리즘 기초

- 두 문자열 연결하기

```
while (b[j] != '\0') {  
    a[i] = b[j];  
    i++;  
    j++;  
}
```

```
a[i] = '\0';  
printf("%s\n", a);
```

```
/*  
    i=5, a[5]=b[0], smartp  
    i=6, a[6]=b[1], smartph  
    i=7, a[7]=b[2], smartpho  
    i=8, a[8]=b[3], smartphon  
    i=9, a[9]=b[4], smartphone  
    i=10, a[10] = '\0'  
*/
```

smart+phone=smartphone  
smartphone



# 알고리즘 기초

- 두 문자열 연결하기 – strcat() 사용

```
/*
   strcat() 함수
   문자열 끝('\0')을 찾아 자동으로 뒤에 붙여주며,
   복사 후 마지막에 '\0'도 추가합니다.
*/
char str1[20] = "smart";
char str2[] = "phone";

strcat(str1, str2);
printf("%s\n", str1);
```



# 알고리즘 기초

- 두 문자열 연결하기 – strcat() 사용

```
/*  
    strcat() 함수  
    문자열 끝('\0')을 찾아 자동으로 뒤에 붙여주며,  
    복사 후 마지막에 '\0'도 추가합니다.  
*/  
char str1[20] = "smart";  
char str2[] = "phone";  
  
strcat(str1, str2);  
printf("%s\n", str1);
```



# 검색 알고리즘

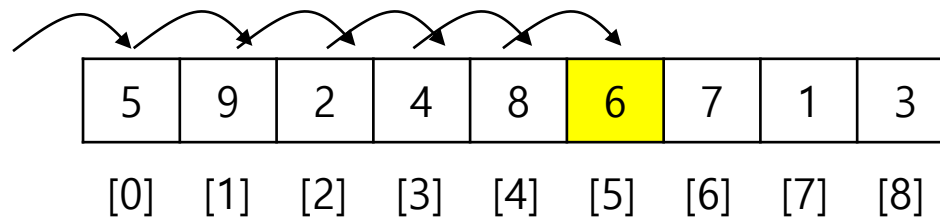
- 순차 검색(sequential search)

- 동작 원리

1. 첫번째 요소부터 하나씩 검사
2. 찾는 값과 같으면 위치 출력
3. 찾았으면 종료
4. 끝까지 못찾으면 "없음" 출력

- 특징

1. 구현이 매우 간단하다.
2. 데이터가 많아지면 속도가 느려진다. – 시간 복잡도  $O(n)$
3. 불필요한 비교 – 값을 찾았어도 반복문이 끝까지 돌



# 검색 알고리즘

- 순차 검색(sequential search)

```
int a[9] = { 5, 9, 2, 4, 8, 6, 7, 1, 3 };
int i;
int x = 6; //찾을 값
int found = 0; //상태(찾음, 못찾음)

for (i = 0; i < 9; i++) {
    if (a[i] == x) {
        printf("%d은 a[%d]에 있습니다.\n", x, i);
        found = 1; //찾음
        break; //더 이상 찾을 필요 없음!!
    }
}

if (!found) {
    printf("%d은 없습니다.\n");
}
```





# 검색 알고리즘

- 순차 검색(sequential search) – 함수로 구현

```
void sequentialSearch(int a[], int n, int x) {  
    int i, found = 0;  
  
    for (i = 0; i < n; i++) {  
        if (a[i] == x) {  
            printf("%d은 a[%d]에 있습니다.\n", x, i);  
            found = 1; //찾음  
            break;    //더 이상 찾을 필요 없음!!  
        }  
    }  
  
    if (!found) {  
        printf("%d은 없습니다.\n");  
    }  
}
```



# 검색 알고리즘

- 순차 검색(sequential search) – 함수로 구현

```
int arr[9] = { 5, 9, 2, 4, 8, 6, 7, 1, 3 };  
int size; //배열의 크기  
int x = 6; //찾을 값  
  
size = sizeof(arr) / sizeof(arr[0]);  
  
//순차 탐색 함수 호출  
sequentialSearch(arr, size, x);
```

6은 a[5]에 있습니다.

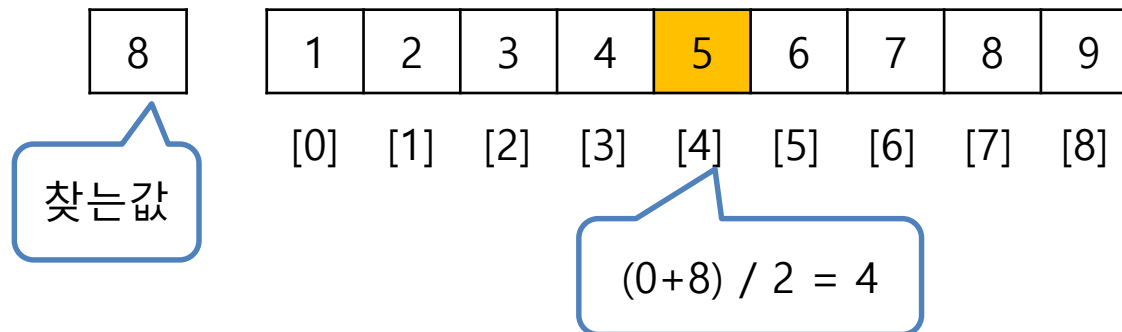


# 검색 알고리즘

- 이분 검색(binary search)

**정렬된** 데이터를 좌우 둘로 나눠서 찾는 값의 검색 범위를 좁혀가는 방법이다.

- 찾을 값 < 가운데 요소 -> 오른쪽 반을 검색 범위에서 제외시킴( $8 < 5$ )
- 찾을 값 > 가운데 요소 -> 왼쪽 반을 검색 범위에서 제외시킴( $8 > 5$ )
- 찾을 값 = 가운데 요소 -> 검색을 완료함



# 검색 알고리즘

- 이분 검색(binary search)

```
int low, high, mid;
int x, found;

//정렬된 배열
int arr[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

low = 0; //첫 인덱스
high = 8; //마지막 인덱스
x = 8; //찾을 값
found = 0; //상태(찾음/못찾음)

while (low <= high) {
    mid = (low + high) / 2; //중간값의 위치
    //printf("%d\n", mid); //4 -> 6 -> 7

    if (arr[mid] == x) {
        printf("%d은 a[%d]에 있습니다.", x, mid);
        found = 1; //찾음
        break;
    }
}
```



# 검색 알고리즘

- 이분 검색(binary search)

```
        else if (arr[mid] < x) {
            low = mid + 1;
        }
        else { //a[mid] > x
            high = mid - 1;
        }
        /*
            mid=4, 5<8, low=5, high=8, mid=6(13/2)
            mid=6, 7<8, low=7, high=8, mid=7(15/2)
            mid=7, 8=8, 찾음
        */
    }

    if (!found) //찾지 못함
        printf("%d은 없습니다.", x);

    return 0;
}
```



# 검색 알고리즘

- 이분 검색(binary search) – 함수로 구현

```
//binarySearch(배열, 배열의 크기, 찾을값)
void binarySearch(int a[], int n, int x) {
    int low, high, mid;
    int found = 0;

    low = 0; //배열의 첫 인덱스
    high = n - 1; //배열의 마지막 인덱스

    while (low <= high) {
        mid = (low + high) / 2; //중간값의 위치
        //printf("%d\n", mid); //4 -> 6 -> 7

        if (a[mid] == x) {
            printf("%d은 a[%d]에 있습니다.", x, mid);
            found = 1;
            break;
        }
    }
}
```



# 검색 알고리즘

- 이분 검색(binary search) – 함수로 구현

```
        else if (a[mid] < x) {  
            low = mid + 1;  
        }  
        else { //a[mid] > x  
            high = mid - 1;  
        }  
    }  
  
    if (!found)  
        printf("%d은 없습니다.", x);  
}
```



# 검색 알고리즘

- 이분 검색(binary search) – 함수로 구현

```
int arr[9] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };  
int size;    //배열의 크기  
int x = 8;   //찾을 값  
  
size = sizeof(arr) / sizeof(arr[0]);  
  
//이분 탐색 함수 호출  
binarySearch(arr, size, x);
```

