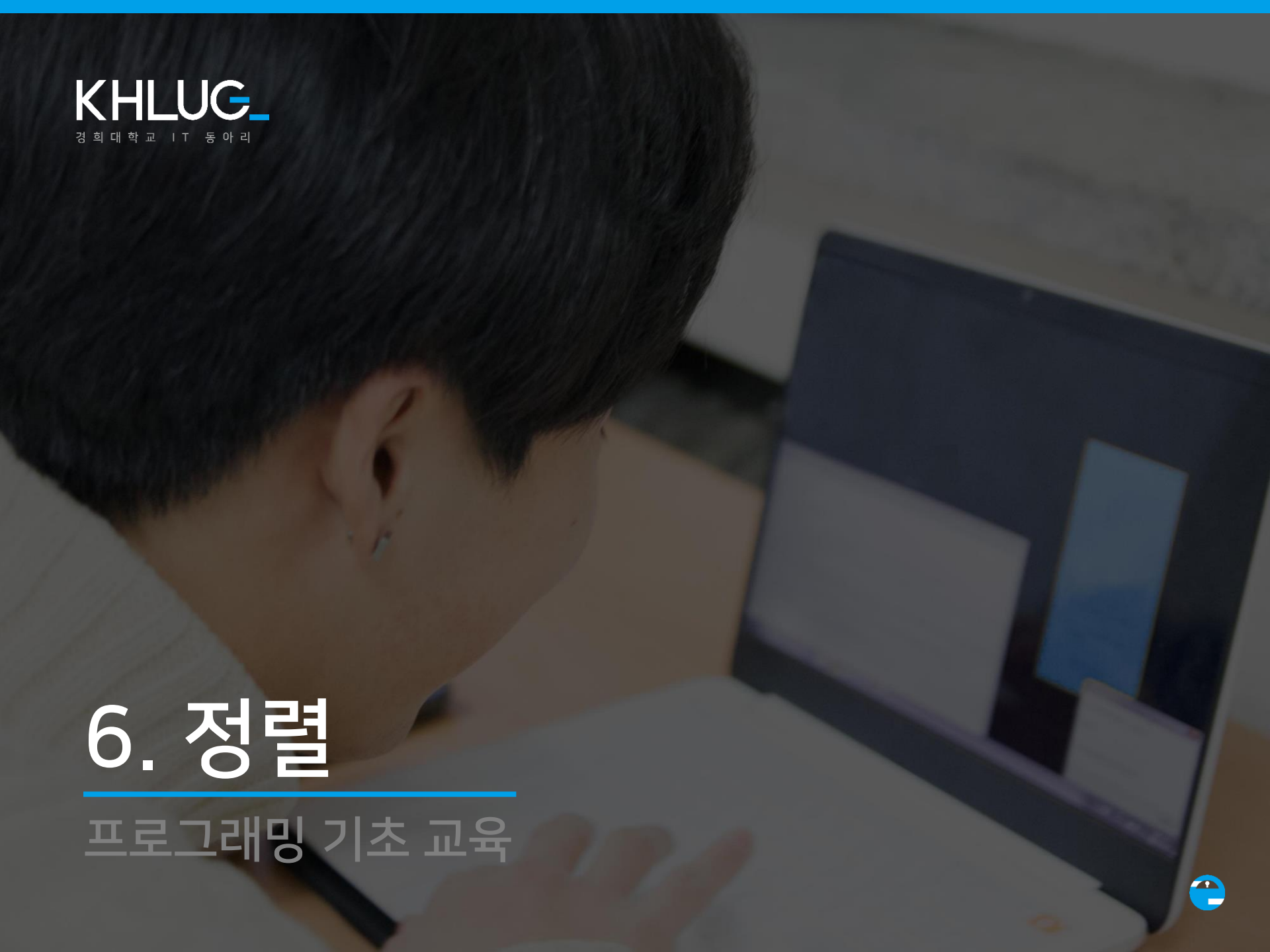


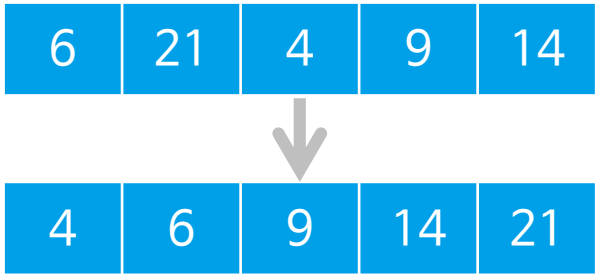
KHLUG
경희대학교 IT 동아리

6. 정렬

프로그래밍 기초 교육

정렬

- 배열 원소를 순서대로 정렬해보자.

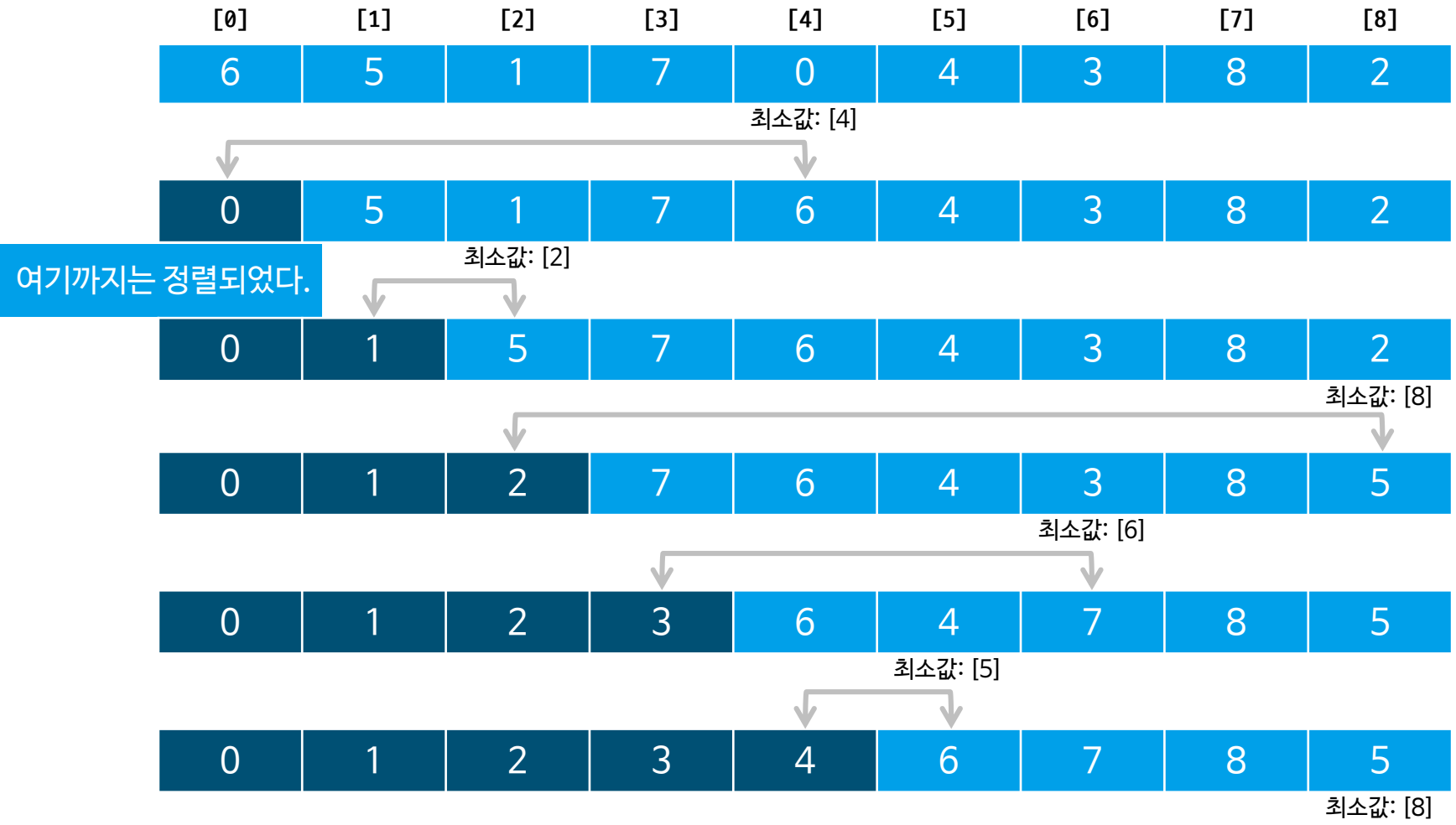


- 앞에서 배운 '자리 바꾸기'를 많이 사용한다.

```
void swap(int& n1, int& n2){
    int temp = n1;
    n1 = n2;
    n2 = temp;
    return;
}
```

선택 정렬

- 선택 정렬(Selection Sort)에선 최소값을 선택해 앞으로 보낸다.

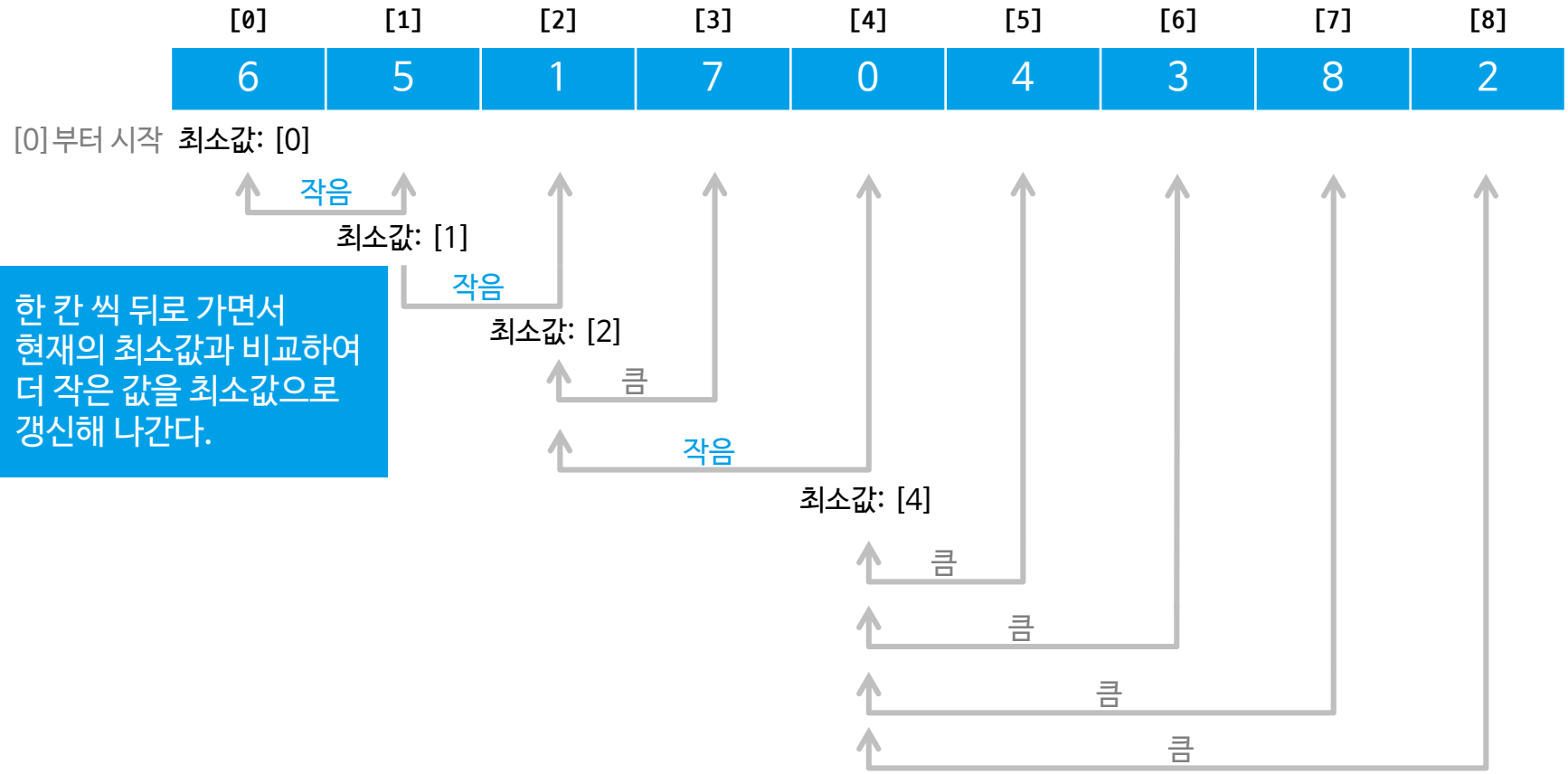


선택 정렬

- 이런 식으로 제일 작은 원소부터 앞으로 보내는 방식의 정렬이다.

선택 정렬

- 가장 작은 값은 어떻게 비교할까?



선택 정렬

```
void SelectionSort(int list[], int n);
// n은 배열의 크기

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

    SelectionSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

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

    // 최소값의 인덱스로 사용할 변수 선언

    // 반복 (마지막에서 두 번째 것까지)

        // 최소값 설정

        // 반복 (다음 것부터 맨 뒤 것까지)

            // 만약 (최소값보다 더 작으면)

                // 최소값 갱신
            // 끝 - 비교
        // 끝 - 반복

        // 최소값 맨 앞으로 보내기

    // 끝 - 반복
}
```

선택 정렬

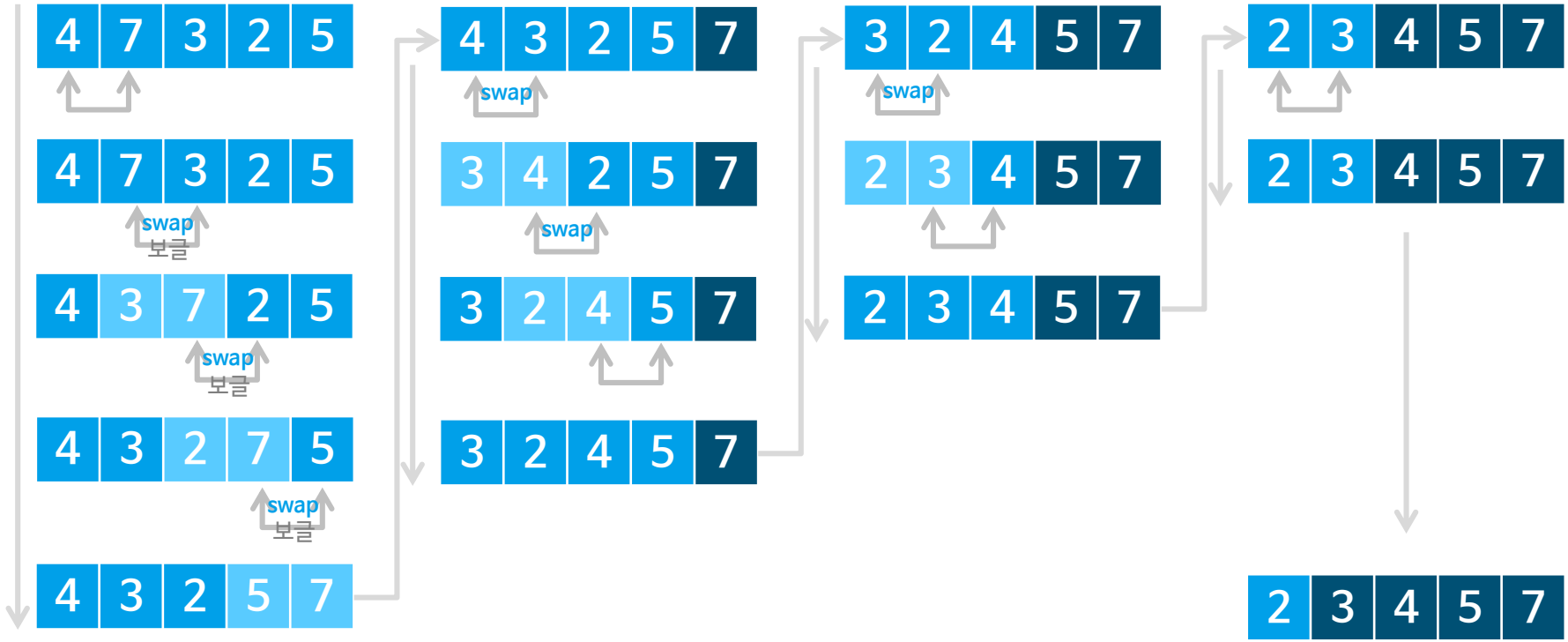
```
void SelectionSort(int list[], int n);
// n은 배열의 크기

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

    SelectionSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

```
void SelectionSort(int list[], int n){
    int min;
    // 최소값의 인덱스로 사용할 변수 선언
    for(int i = 0; i < n-1; i++){
    // 반복 (마지막에서 두 번째 것까지)
        min = i;
        // 최소값 설정
        for(int j = i+1; j < n; j++){
        // 반복 (다음 것부터 맨 뒤 것까지)
            if(list[j] < list[min]){
            // 만약 (최소값보다 더 작으면)
                min = j;
                // 최소값 갱신
            } // 끝 - 비교
        } // 끝 - 반복
        swap(list[i], list[min]);
        // 최소값 맨 앞으로 보내기

    } // 끝 - 반복
}
```

이곳에 배열 값을 출력하는 코드를 넣어 정렬이 어떤 과정으로 이루어지는지 확인해보자.

거품 정렬

- 거품 정렬(Bubble Sort)에서는 인접한 원소끼리 비교하며 정렬한다.
  큰 원소부터 거품이 수면으로 떠오르는 모양이다.



거품 정렬

```
void BubbleSort(int list[], int n);
// n은 배열의 크기

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

    BubbleSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

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

    // 반복 (전부 다 확인)

        // 반복 (자기 빼고 다 확인)

            // 만약 (앞보다 뒤가 작으면)

                // (서로 바꿈)
            // 끝 - 비교
        // 끝 - 반복
    // 끝 - 반복
}
```

거품 정렬

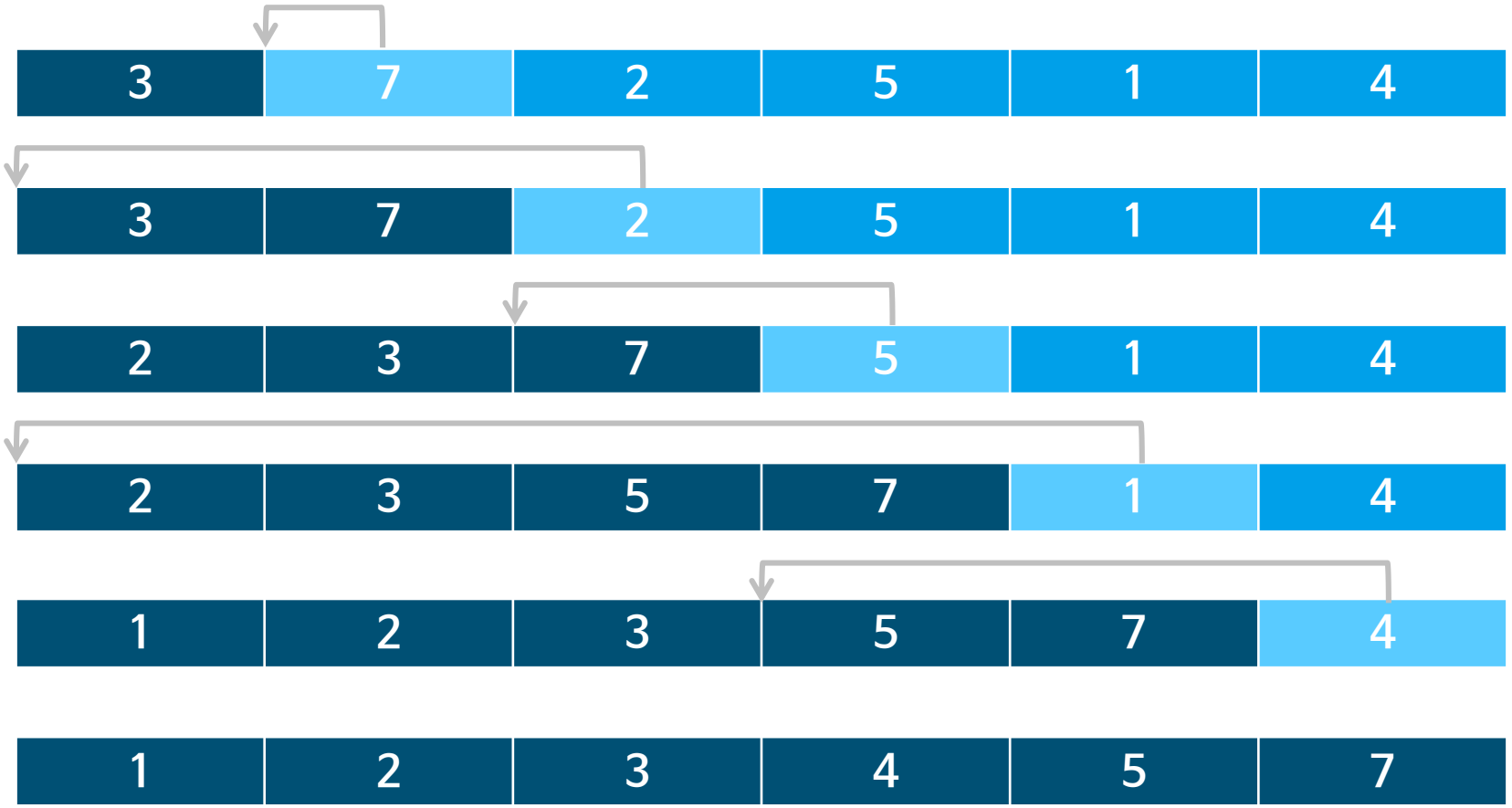
```
void BubbleSort(int list[], int n);
// n은 배열의 크기

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

    BubbleSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

```
void BubbleSort(int list[], int n){
    for(int i = 0; i < n; i++){
    // 반복 (전부 다 확인)
        for(int i = 0; j < n-i-1; j++){
        // 반복 (자기 빼고 다 확인)
            if(list[j] > list[j+1]){
            // 만약 (앞보다 뒤가 작으면)
                swap(list[j], list[j+1]);
                // (서로 바꿈)
            } // 끝 - 비교

        } // 끝 - 반복
    } // 끝 - 반복
}
```

이곳에 배열 값을 출력하는 코드를 넣어 정렬이 어떤 과정으로 이루어지는지 확인해보자.

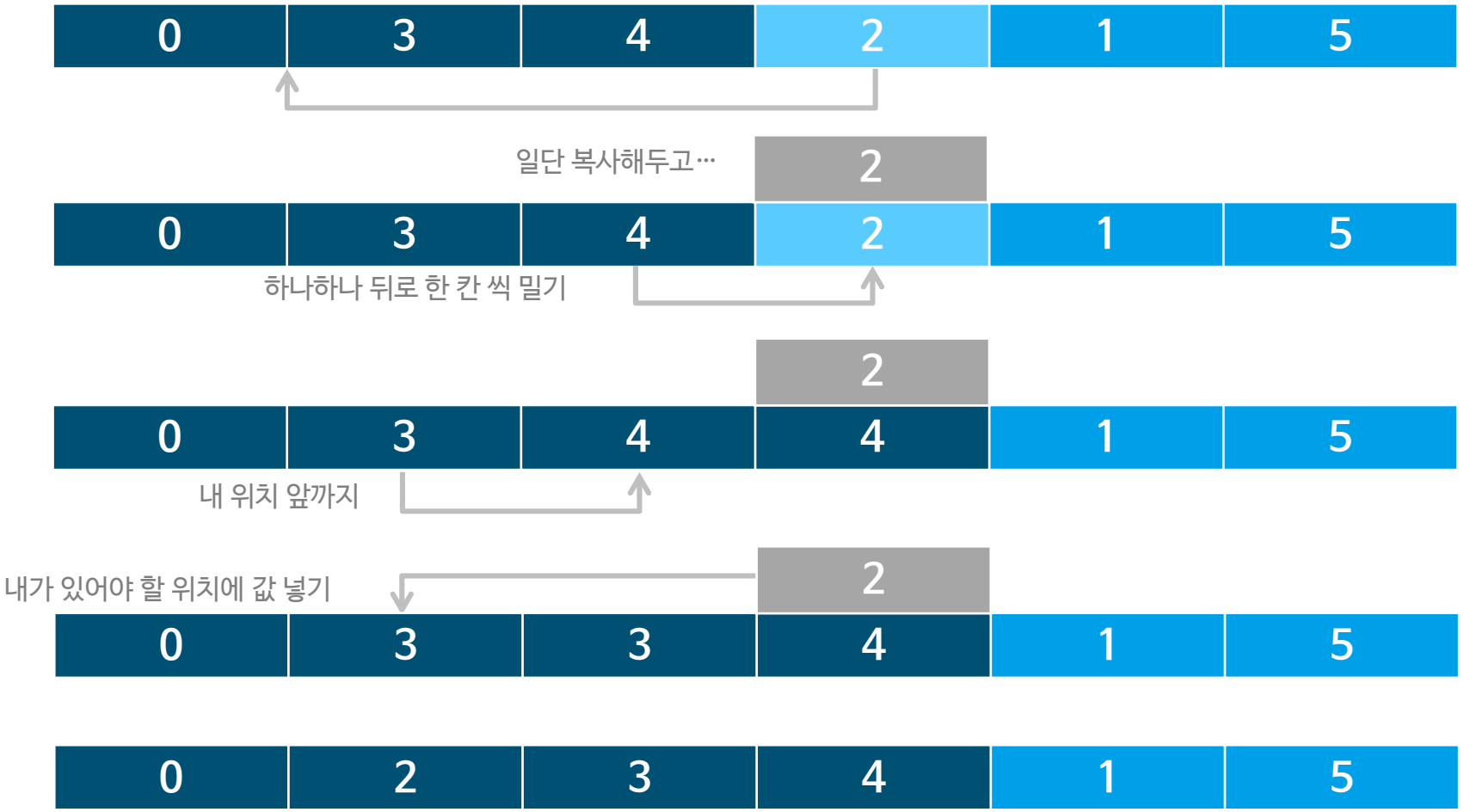
삽입 정렬

- 삽입 정렬(Insertion Sort)는 이미 정렬된 부분과 비교하여 자신의 위치를 찾아 삽입되어가며 정렬한다.



삽입 정렬

- 삽입은 어떻게 할까?



삽입 정렬

```
void InsertionSort(int list[], int n);
// n은 배열의 크기

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

    InsertionSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

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

    // 삽입 시 필요한 임시 변수 선언

    // 반복 (모든 원소를 확인)

        // 임시 변수 초기화 (넣고 싶은 값)

        // 반복 (정렬된 배열의 뒤부터 확인)

            // 만약 (비교하는 값이 더 크면)

                // 한 칸씩 뒤로 밀기

            // 아니면

                // 여기가 내 위치이므로 삽입

                // 더 이상 살펴보지 않음
            // 끝 - 비교
        // 끝 - 반복
    // 끝 - 반복
}
```

삽입 정렬

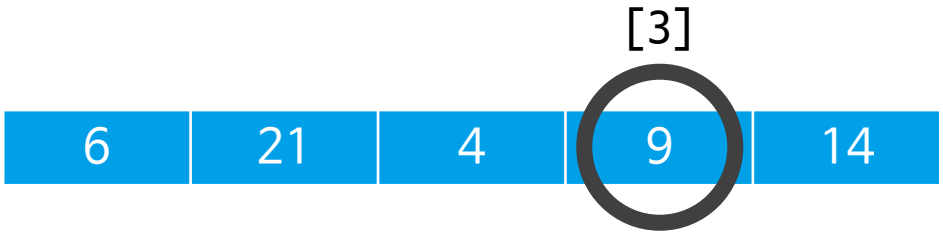
```
void InsertionSort(int list[], int n);
// n은 배열의 크기

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

    InsertionSort(list, 9);
    // 선택정렬 실행

    for(int i=0; i<9; i++)
        cout << list[i] << ' ';
        // 배열 값 출력

    return 0;
}
```

```
void InsertionSort(int list[], int n){
    int target;
    // 삽입 시 필요한 임시 변수 선언
    for(int i = 0; i < n; i++){
    // 반복 (모든 원소를 확인)
        target = list[i];
        // 임시 변수 초기화 (넣고 싶은 값)
        for(int j = i-1; j >= 0; j--){
        // 반복 (정렬된 배열의 뒤부터 확인)
            if(list[j] > target){
            // 만약 (비교하는 값이 더 크면)
                list[j+1] = list[j];
                // 한 칸씩 뒤로 밀기
            }else{
            // 어라, 얘부터 나보다 작네
                list[j] = target;
                // 그럼 내 위치는 여기!
                break;
                // 더 볼 필요 없다. 탈출!
            } // 끝 - 비교
        } // 끝 - 반복
    } // 끝 - 반복
}
```

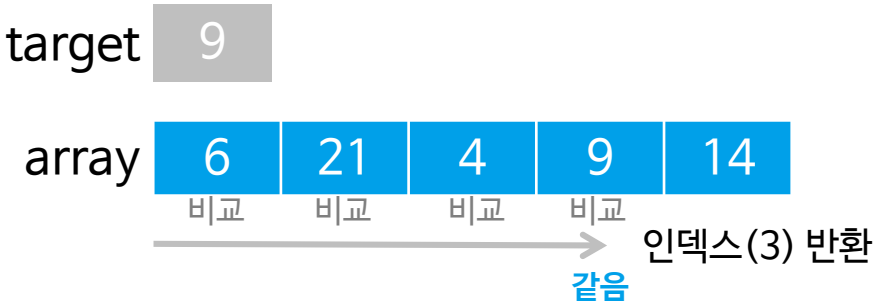
이곳에 배열 값을 출력하는 코드를 넣어 정렬이 어떤 과정으로 이루어지는지 확인해보자.

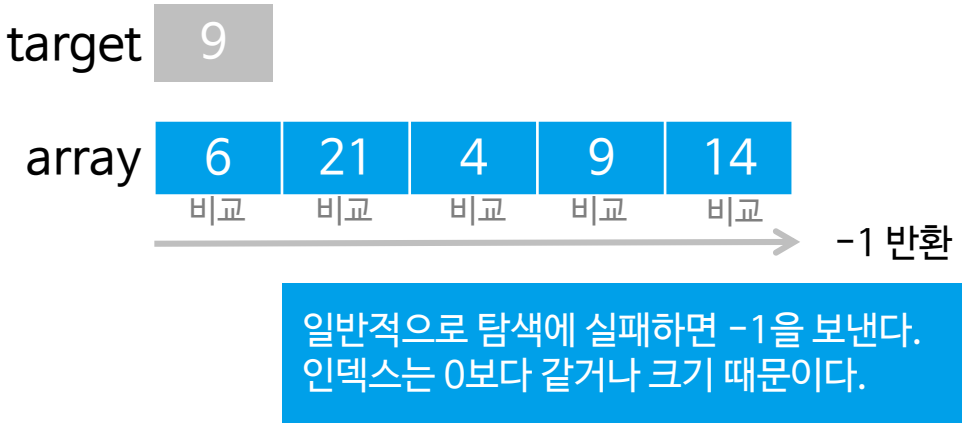
선형 탐색

- 탐색(Search)은 배열에 원하는 값이 있는지 찾는 것이다.



- 선형 탐색(Linear Search) 또는 순차 탐색(Sequential Search)은 배열의 처음부터 끝까지 순서대로 찾아보는 것이다.





일반적으로 탐색에 실패하면 -1을 보낸다.
인덱스는 0보다 같거나 크기 때문이다.

선형 탐색

```
int SequentialSearch(int list[],
    int length, int target);
// length는 배열의 크기

int main(){
    int list[9] = {6,5,1,7,0,4,3,8,2};
    cout << SequentialSearch(list, 7)
        << endl;
    cout << SequentialSearch(list, 8)
        << endl;
    cout << SequentialSearch(list, 9)
        << endl;

    return 0;
}
```

```
int SequentialSearch(int list[],
    int length, int target){


    // 반복 (처음부터 끝까지)

        // 만약 (target과 값이 같으면)

            // 인덱스 반환
        // 끝 - 비교
    // 끝 - 반복


    // 반복문을 빠져 나왔다면 -1 반환
}
```

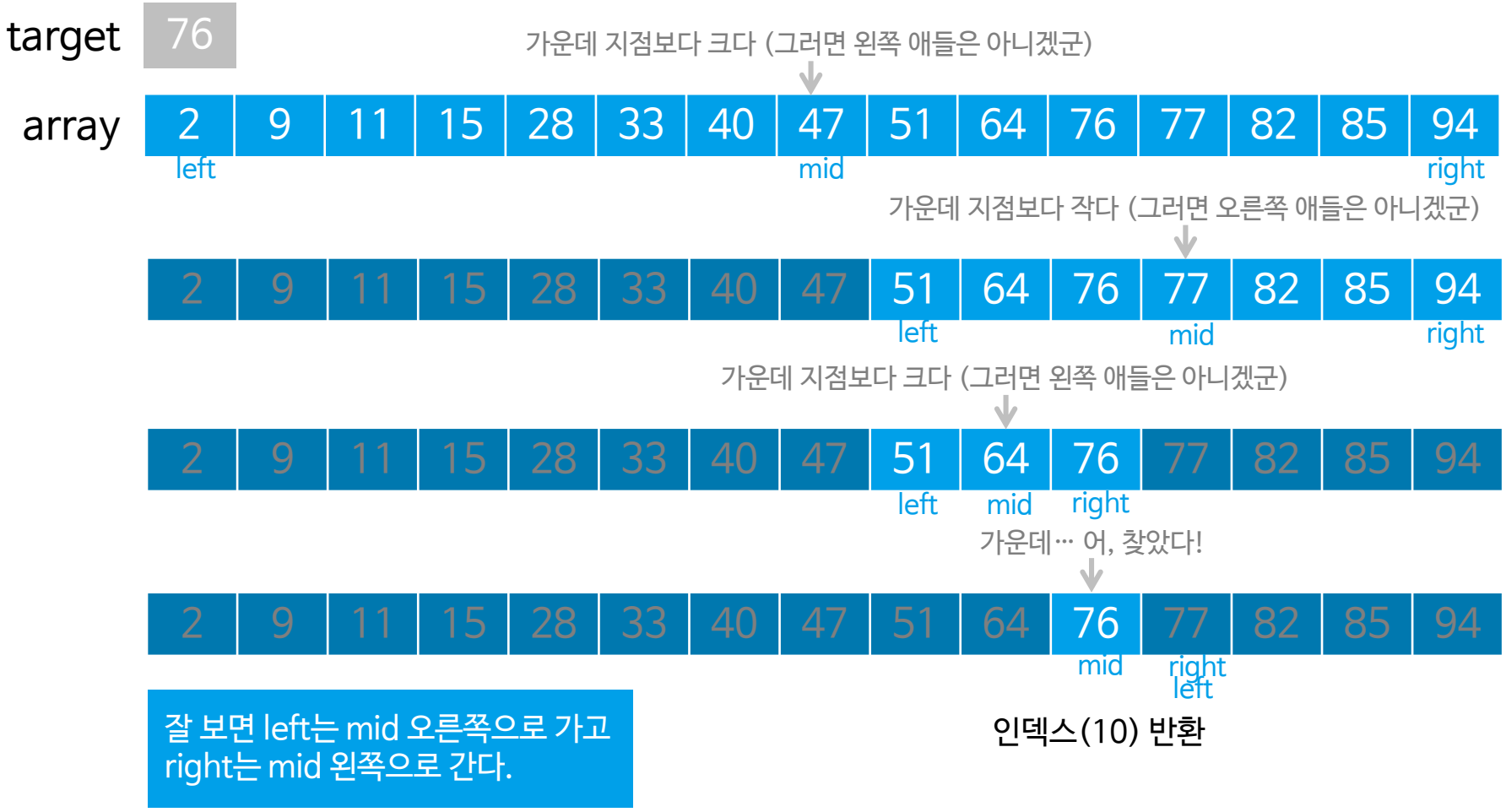
선형 탐색

```
int SequentialSearch(int list[],
    int length, int target);
// length는 배열의 크기

int main(){
    int list[9] = {6,5,1,7,0,4,3,8,2};
    cout << SequentialSearch(list, 7)
        << endl;
    cout << SequentialSearch(list, 8)
        << endl;
    cout << SequentialSearch(list, 9)
        << endl;

    return 0;
}
```

```
int SequentialSearch(int list[],
    int length, int target){

    for(int i = 0; i < length; i++){
    // 반복 (처음부터 끝까지)
        if(list[i] == target){
        // 만약 (target과 값이 같으면)
            return i;
            // 인덱스 반환
        } // 끝 - 비교
    } // 끝 - 반복

    return -1;
    // 반복문을 빠져 나왔다면 -1 반환
}
```

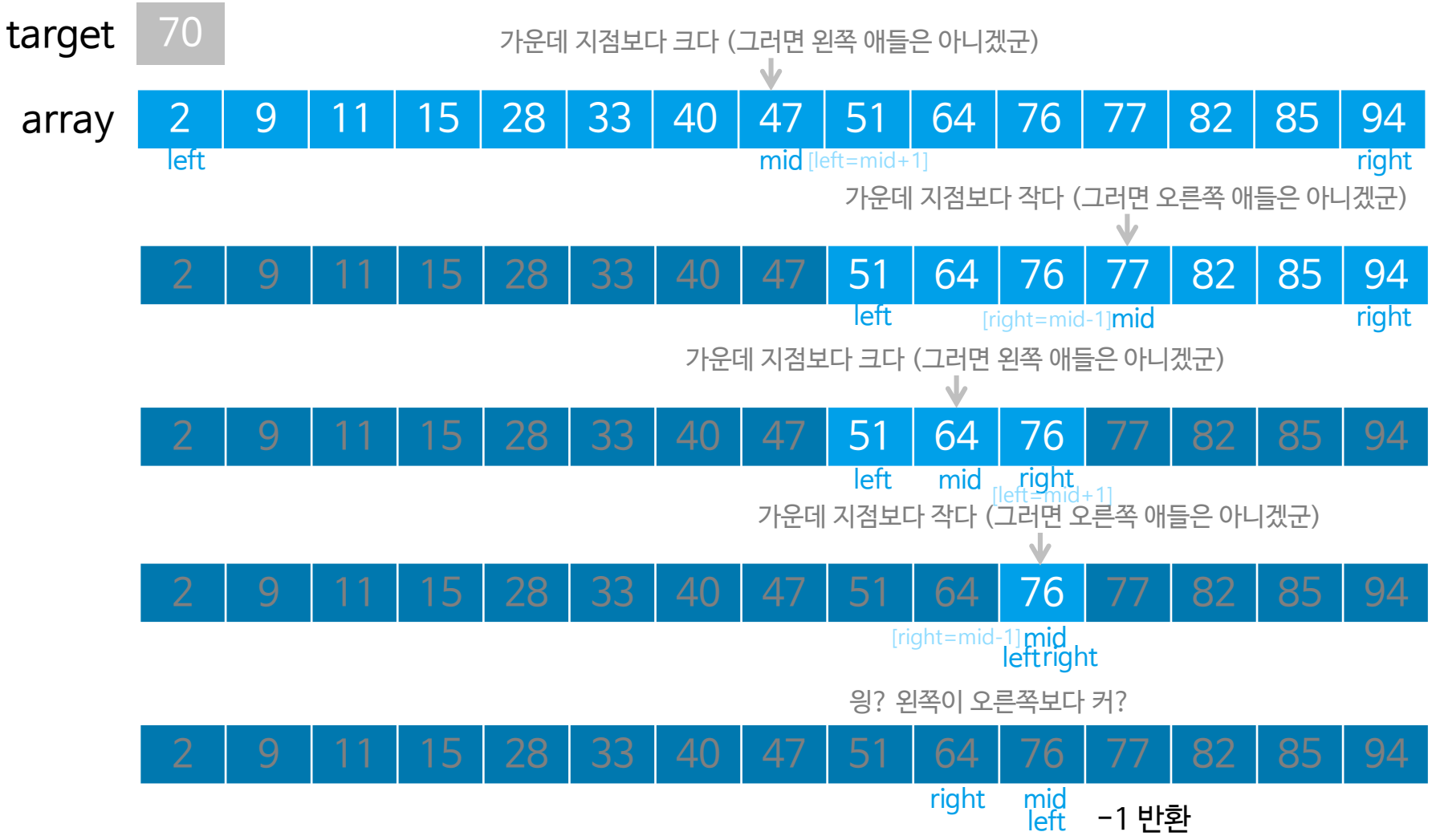
이진 탐색

- 이진 탐색(Binary Search)은 정렬된 배열에서 탐색을 한다.



이진 탐색

- 배열에 원하는 값이 없으면 아래와 같이 될 것이다.



이진 탐색

```
int BinarySearch(int list[], int length,
    int target);

int main(){
    int list[9] = {6,5,1,7,0,4,3,8,2};
    Sort(list, 9);
    // 정렬 실행 (종류는 아무거나 하자)

    int find;
    cin >> find;
    // 탐색 대상 입력
    int result = BinarySearch(list,9,find);
    // 이진탐색 결과 저장

    if(result==-1)
        cout << "… 찾을 수 없다 …" << endl;
    else
        cout << result << "번째에 있음!" << endl;

    return 0;
}
```

〈algorithm〉 라이브러리의 sort()를 사용할 수 있다.

```
int BinarySearch(int list[], int length,
    int target){

    // 왼쪽 끝 선언, 초기화
    // 오른쪽 끝 선언, 초기화
    // 가운데 선언


    // 반복 (왼쪽이 오른쪽보다 작을 때)

        // 가운데 초기화

        // 만약 (가운데 값이 탐색 대상과 같으면)

            // 해당 인덱스 반환

        // 아닌데 만약 (탐색 대상이 더 크면)

            // 왼쪽은 아니므로 왼쪽 끝 갱신

        // 아닌데 만약 (탐색 대상이 더 작으면)

            // 오른쪽은 아니므로 오른쪽 끝 갱신
        // 끝 - 비교
    // 끝 - 반복

    // 반복문을 빠져 나왔다면 -1 반환
}
```

이진 탐색

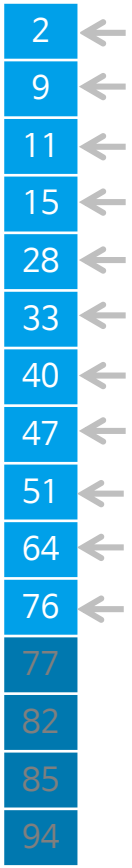
```
int BinarySearch(int list[], int length,
    int target);

int main(){
    int list[9] = {6,5,1,7,0,4,3,8,2};
    Sort(list, 9);
    // 정렬 실행 (종류는 아무거나 하자)

    int find;
    cin >> find;
    // 탐색 대상 입력
    int result = BinarySearch(list,9,find);
    // 이진탐색 결과 저장

    if(result==-1)
        cout << "… 찾을 수 없다 …" << endl;
    else
        cout << result << "번째에 있음!" << endl;

    return 0;
}
```

```
int BinarySearch(int list[], int length,
    int target){

    int left = 0; // 왼쪽 끝 선언, 초기화
    int right = length-1; // 오른쪽 끝 선언, 초기화
    int mid; // 가운데 선언

    while(left <= right){
    // 반복 (왼쪽이 오른쪽보다 작을 때)
        mid = left + (left+right)/2
        // 가운데 초기화
        if(list[mid] == target){
        // 만약 (가운데 값이 탐색 대상과 같으면)
            return mid;
            // 해당 인덱스 반환
        }else if(list[mid] < target){
        // 아닌데 만약 (탐색 대상이 더 크면)
            left = mid+1;
            // 왼쪽은 아니므로 왼쪽 끝 갱신
        }else if(list[mid] > target){
        // 아닌데 만약 (탐색 대상이 더 작으면)
            right = mid-1;
            // 오른쪽은 아니므로 오른쪽 끝 갱신
        } // 끝 - 비교
    } // 끝 - 반복
    return -1;
    // 반복문을 빠져 나왔다면 -1 반환
}
```

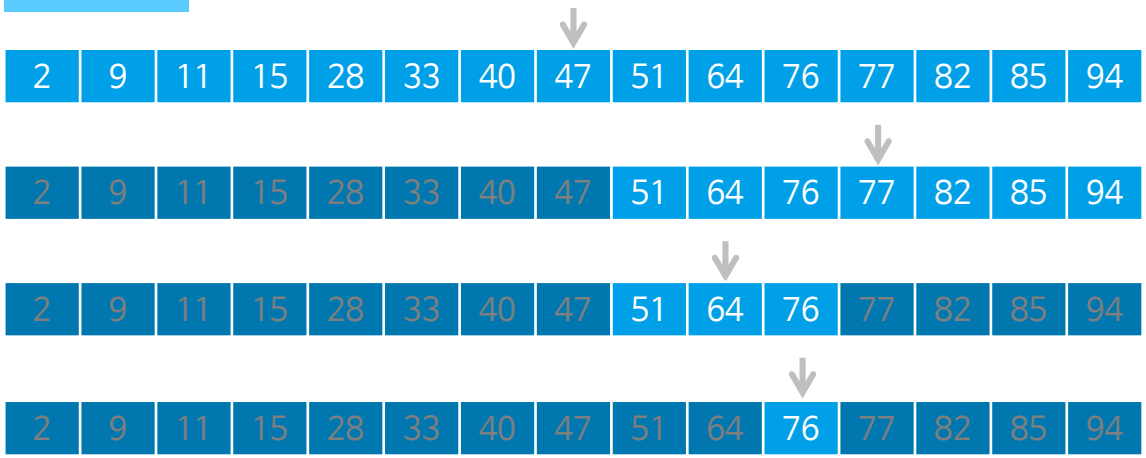
누가 더 빠를까

target 76

선형 탐색



이진 탐색



선형 탐색은 모든 원소와 비교해야 돼서 11번이나 비교를 하는 반면 이진 탐색에서는 4번의 비교로 찾아낸다.

- 배열의 길이가 n일 때 선형 탐색은 실패할 때까지 최대 n번의 비교를 하지만 이진 탐색은 $\log_2 n$번으로 더 빨리 끝난다.

$n=2^a$일 때 $a=\log_2 n$이다.

2017.04.11. 프로그래밍 기초 (2017-1)
with D.com