재귀함수&정렬

목차

선택정렬

1. 주어진 리스트에서 최솟값을 찾는다.

2. 최솟값을 맨 앞 자리의 값과 교환한다.

3. 맨 앞 자리를 제외한 나머지 값들 중 최솟값을 찾아 위와 같은 방법으로 반복한다.

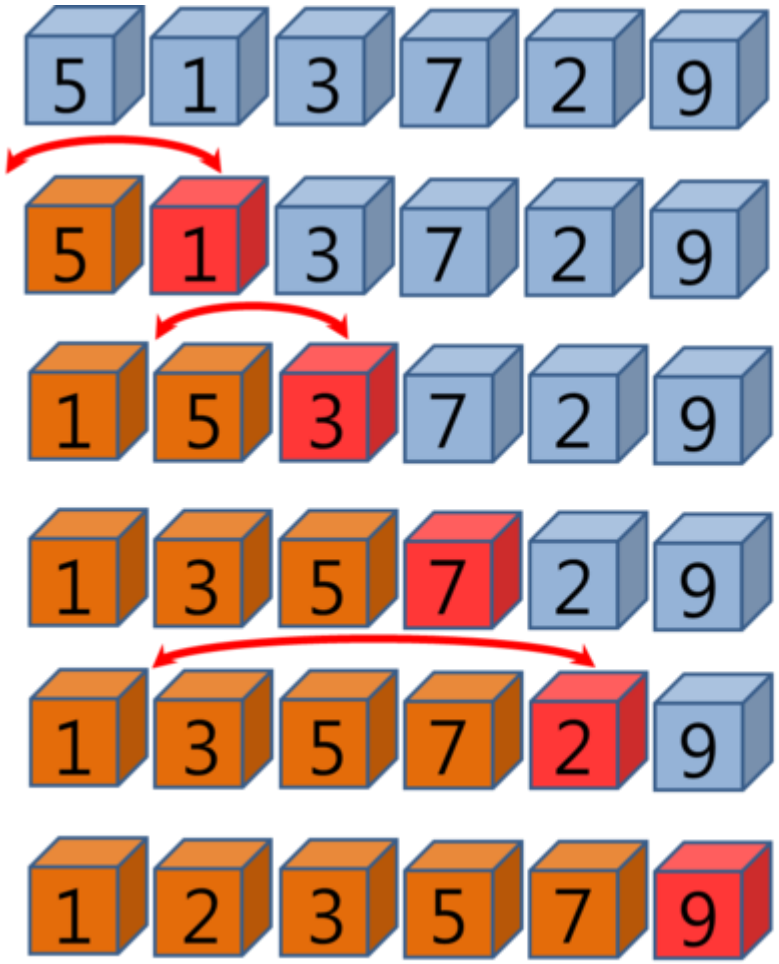
```c
int main(void){
    int i, j, min, index, temp;
    int array[10] = {1,10,5,8,7,6,4,3,2,9};
    for(i=0; i< 10; i++){
        min =9999;
        for(j=i; j<10; j++){ //for문 돌리면서 최소값 찾으면 min 값 바꾸기
            if(min>array[j]){
                min = array[j];
                index =j;
            }
        }
        //temp 이용해서 서로 교환
        temp = array[i];
        array[i] = array[index];
        array[index] = temp;
    }
}

```

단점

- 불안정 정렬이다
- **[B1, B2, C, A] (A < B < C)**
- B1이 B2보다 크거나 작은 것이 아니**다**

- 그럼 순서대로 순회하면서 교환한다면 이렇다.
- round 1 : [**A**, B2, C, **B1**]
- round 2 : [A, **B2**, C, B1]
- round 3 : [A, B2, **B1**, **C**]
-
- 이렇게 초기의 B1 B2의 순서가 뒤 바뀐 것을 볼 수 있다.

삽입정렬



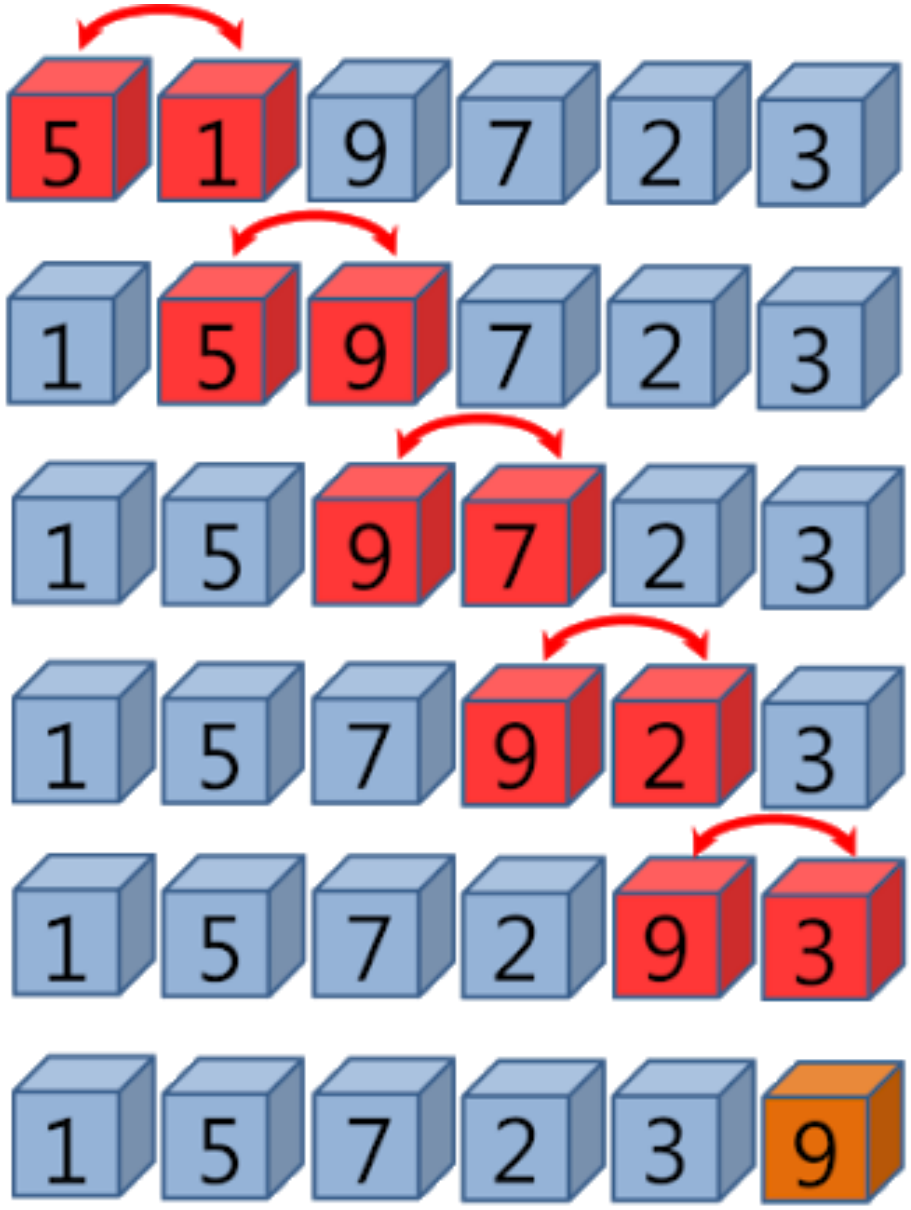
1. 현재 타겟이 되는 숫자와 이전 위치에 있는 원소들을 비교한다.
(첫 번째 타겟은 두 번째 원소부터 시작한다.)

2. 타겟이 되는 숫자가 이전 위치에 있던 원소보다 작다면 위치를 서로 교환한다.

3. 그 다음 타겟을 찾아 위와 같은 방법으로 반복한다.

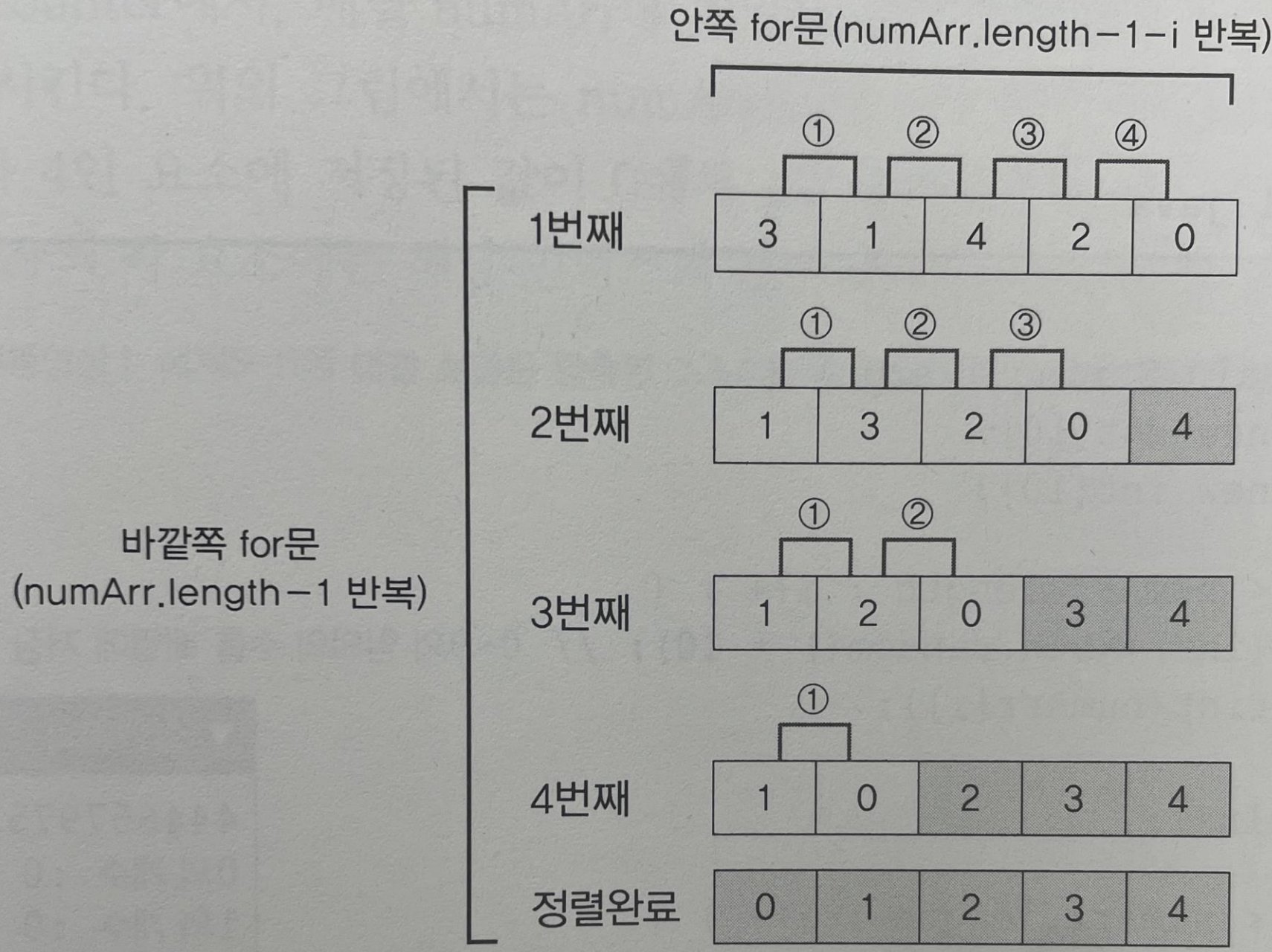
```c
int main(void){
    int i, j, temp;
    int array[10] = {1,10,5,8,7,6,4,3,2,9};
    for(i=0; i< 9; i++){ //
        j=i;
        while(array[j]>array[j+1]){ //왼쪽에 있는 값이 오른쪽에 있는 값보다 크다면
            //위치 바꿈
            temp=array[j];
            array[j] = array[j+1];
            array[j+1]=temp;
            j--;
        }


}

```

버블 정렬



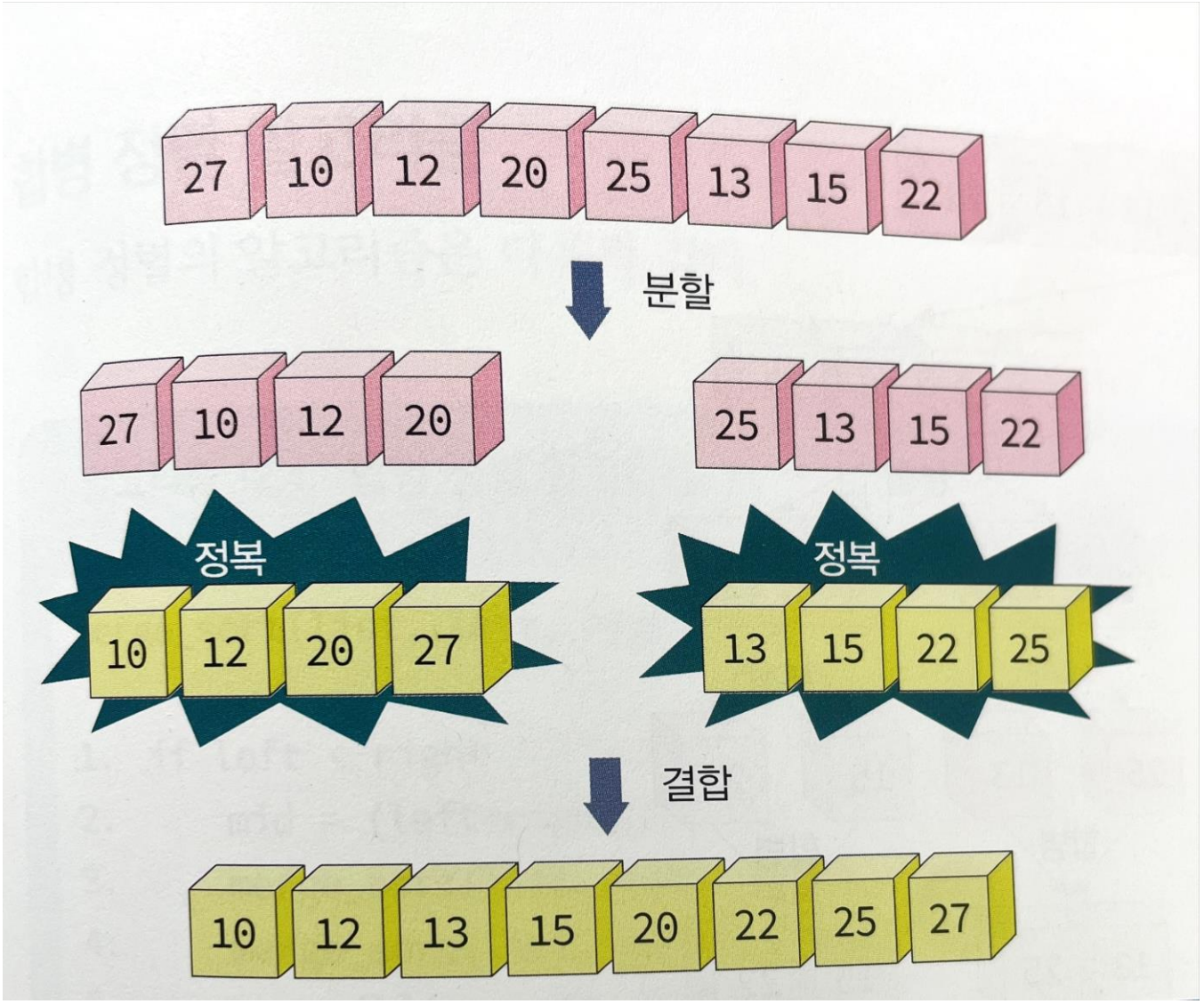
1. 앞에서부터 현재 원소와 바로 다음의 원소를 비교한다.
2. 현재 원소가 다음 원소보다 크면 원소를 교환한다.
3. 다음 원소로 이동하여 해당 원소와 그 다음원소를 비교한다.

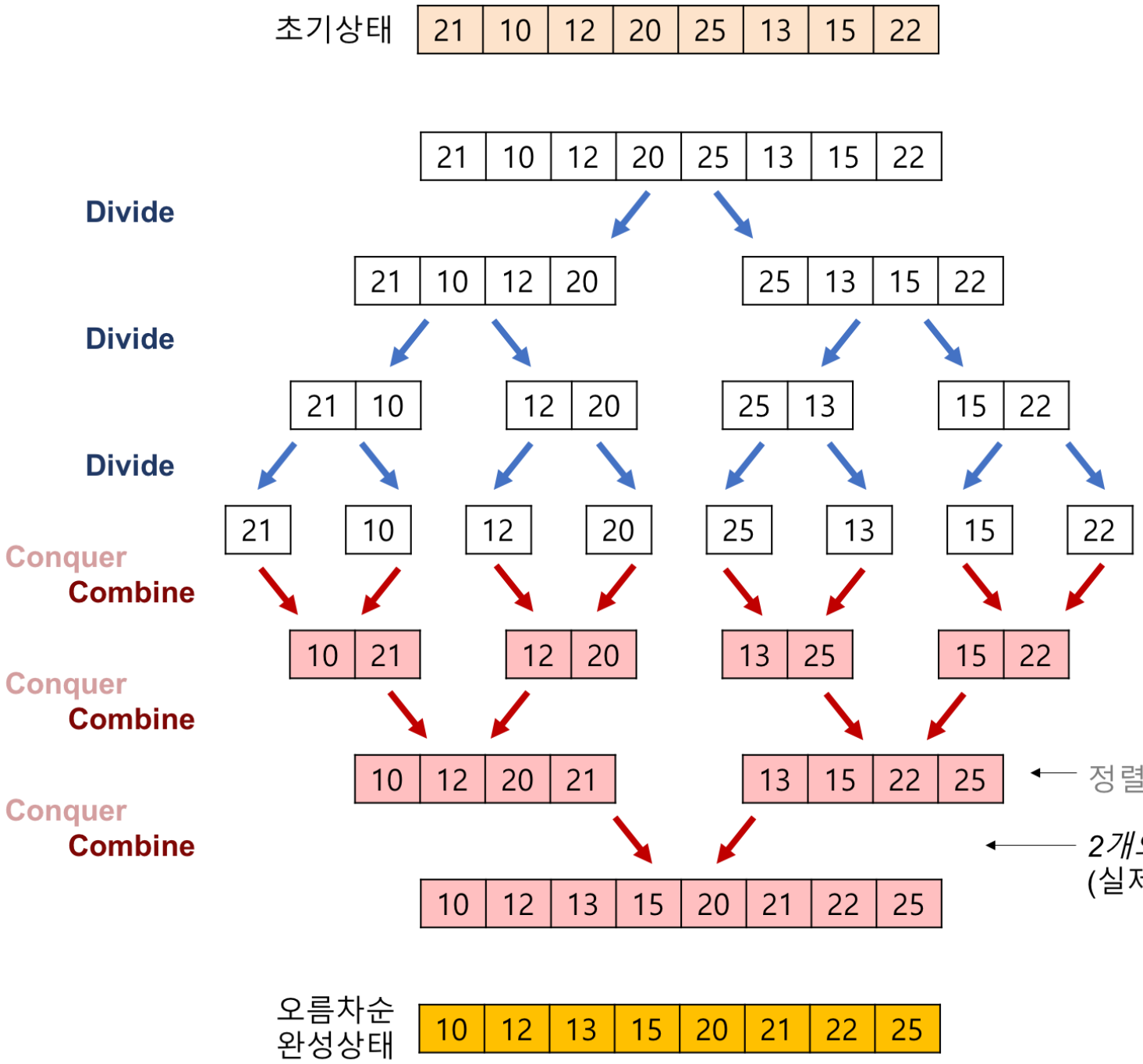
교해야 한다.


안쪽 for문(numArr.length−1−i 반복)
바깥쪽 for문
(numArr.length−1 반복)
1번째
2번째
3번째
4번째
정렬완료
① ② ③ ④
3 1 4 2 0
① ② ③
1 3 2 0 4
① ②
1 2 0 3 4
①
1 0 2 3 4
0 1 2 3 4

```
int main(void) {
    int i, j, temp;
    int array[10] = {1, 10, 5, 8, 7, 6, 4, 3, 2, 9};
    for(i = 0; i < array.length-1; i++) {
        for(j = 0; j < array.length -1 -i; j++) {
            if(array[j] > array[j + 1]) {
                temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
    return 0;
}
```

병합 정렬


27 10 12 20 25 13 15 22
분할
27 10 12 20
25 13 15 22
정복
10 12 20 27
정복
13 15 22 25
결합
10 12 13 15 20 22 25 27

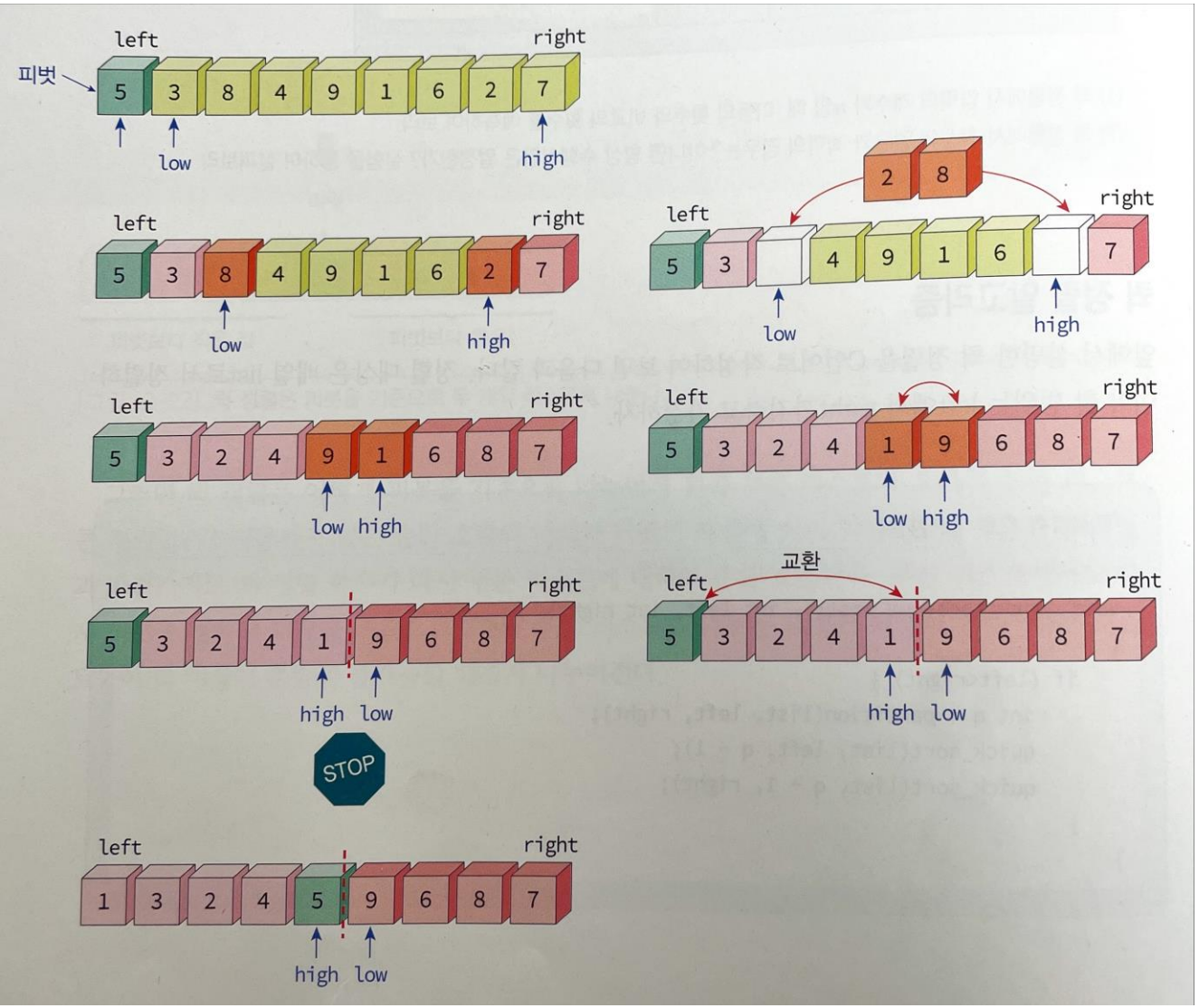


1. **주어진 리스트를 절반으로 분할하여 부분리스트로 나눈다. (Divide : 분할)**

2. **해당 부분리스트의 길이가 1이 아니라면 1번 과정을 되풀이한다.**

3. **인접한 부분리스트끼리 정렬하여 합친다. (Conqure : 정복)**

퀵 정렬



1. 피벗을 하나 선택한다.
2. 피벗을 기준으로 양쪽에서 피벗보다 큰 값, 혹은 작은 값을 찾는다. 왼쪽에서부터는 피벗보다 큰 값을 찾고, 오른쪽에서부터는 피벗보다 작은 값을 찾는다.

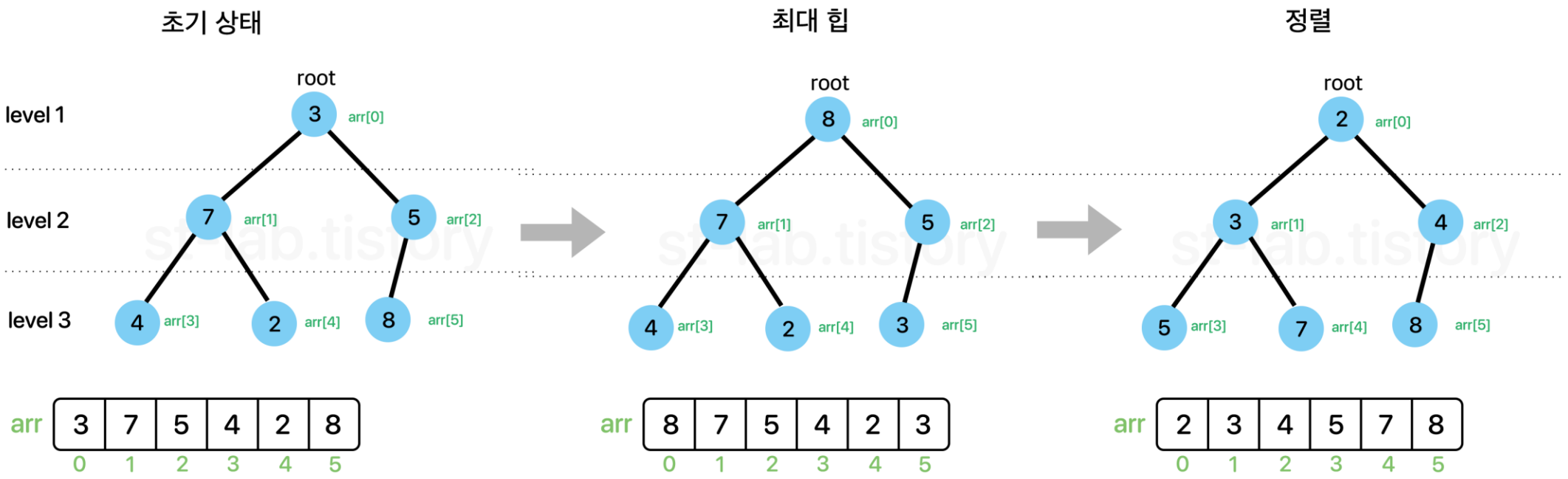
3. 양 방향에서 찾은 두 원소를 교환한다.

4. 왼쪽에서 탐색하는 위치와 오른쪽에서 탐색하는 위치가 엇갈리지 않을 때 까지 2번으로 돌아가 위 과정을 반복한다.

5. 엇갈린 기점을 기준으로 두 개의 부분리스트로 나누어 1번으로 돌아가 해당 부분리스트의 길이가 1이 아닐 때 까지 1번 과정을 반복한다. (Divide : 분할)

6. 인접한 부분리스트끼리 합친다. (Conqure : 정복)

힙 정렬



정렬해야 할 n개의 요소들로 최대 힙(완전 이진 트리 형태)을 만든다.

그 다음으로 한 번에 하나씩 요소를 힙에서 꺼내서 배열의 뒤부터 저장하면 된다.

삭제되는 요소들(최댓값부터 삭제)은 값이 감소되는 순서로 정렬되게 된다.

시간복잡도 비교

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셸 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n\log_2 n$	$n\log_2 n$	n^2	0.014
힙 정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.034
병합정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.026

단순(구현 간단)하지만 비효율적인 방법: 삽입 정렬, 선택 정렬, 버블 정렬
복잡하지만 효율적인 방법: 퀵 정렬, 힙 정렬, 합병 정렬, 기수 정렬