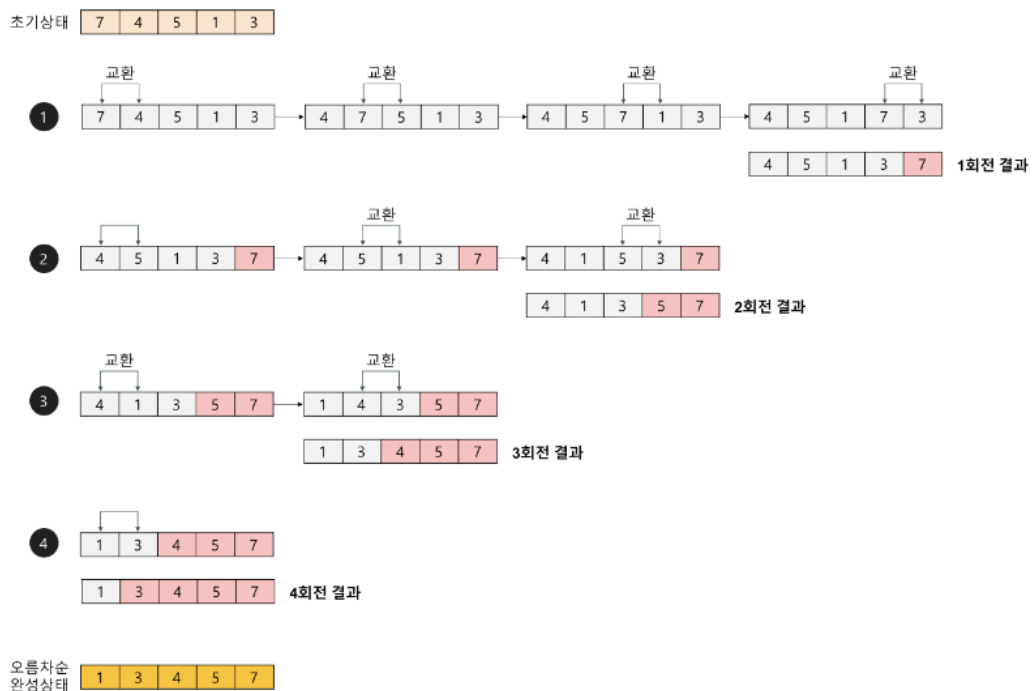


# 알고리즘 스터디 2주차 주제

## 정렬

### 1) 버블 정렬

- 서로 인접한 두 원소를 검사하여 정렬하는 알고리즘
  - 인접한 2개의 레코드를 비교하여 크기가 순서대로 되어 있지 않으면 서로 교환한다.
  - 첫번째 인덱스와 두번째 인덱스, 두번째 인덱스와 세번째 인덱스, 세번째 인덱스와 네번째 인덱스,,, 이렇게 (n-1)번째 인덱스와 n번째 인덱스와 비교하여 교환하면서 정렬한다.
- 예시



- 코드

```

void bubble_sort(int list[], int n){
    int i, j, temp;

    for(i=n-1; i>0; i--){
        // 0 ~ (i-1)까지 반복
        for(j=0; j<i; j++){
            // j번째와 j+1번째의 요소가 크기 순이 아니면 교환
            if(list[j]>list[j+1]){
                temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
            }
        }
    }
}

```

## 2) 선택 정렬


- 해당 순서에 원소를 넣을 위치는 이미 정해져 있고, 어떤 원소를 넣을지 선택하는 알고리즘
  - 첫번째 인덱스에는 가장 최솟값을 넣는다.
  - 두번째 인덱스에는 남은 값 중에서 가장 최솟값을 넣는다.
  - ...
- 예시

초기상태 

9	6	7	3	5
---	---	---	---	---

① 

9	6	7	3	5
---	---	---	---	---


  

 최솟값 탐색: 3  
 첫 번째 값 9와 최솟값 3을 교환

3	6	7	9	5
---	---	---	---	---

 1회전 결과

② 

3	6	7	9	5
---	---	---	---	---


  

 최솟값 탐색: 5  
 두 번째 값 6과 최솟값 5를 교환

3	5	7	9	6
---	---	---	---	---

 2회전 결과

③ 

3	5	7	9	6
---	---	---	---	---


  

 최솟값 탐색: 6  
 세 번째 값 7과 최솟값 6을 교환

3	5	6	9	7
---	---	---	---	---

 3회전 결과

④ 

3	5	6	9	7
---	---	---	---	---

  

 최솟값 탐색: 7  
 네 번째 값 9와 최솟값 7을 교환

3	5	6	7	9
---	---	---	---	---

 4회전 결과

오름차순  
완성상태 

3	5	6	7	9
---	---	---	---	---

- 코드

```
// 선택 정렬
void selection_sort(int list[], int n){
    int i, j, least, temp;

    // 마지막 숫자는 자동으로 정렬되기 때문에 (숫자 개수-1) 만큼 반복한다.
    for(i=0; i<n-1; i++){
        least = i;

        // 최솟값을 탐색한다.
        for(j=i+1; j<n; j++){
            if(list[j]<list[least])
                least = j;
        }

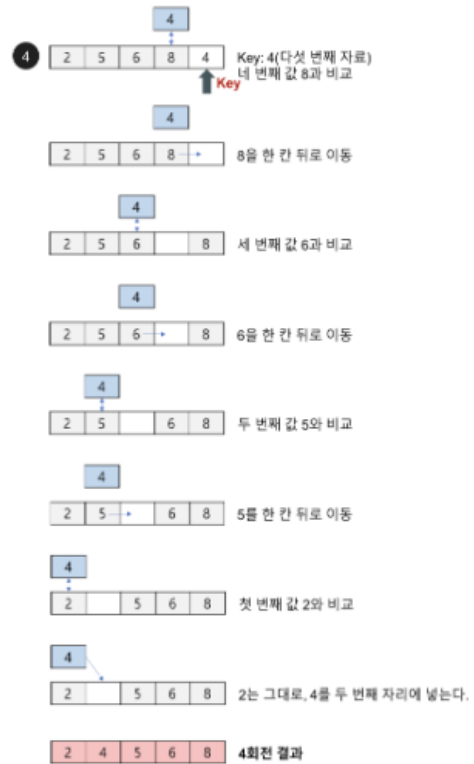
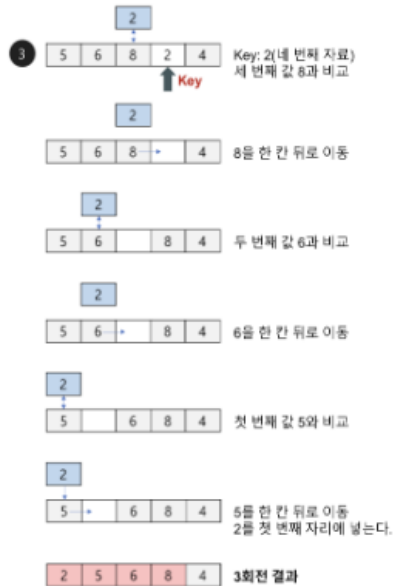
        // 최솟값이 자기 자신이면 자료 이동을 하지 않는다.
        if(i != least){
            SWAP(list[i], list[least], temp);
        }
    }
}
```

### 3) 삽입 정렬

- 앞에서부터 차례대로 이미 정렬된 배열 부분과 비교하여, 자신의 위치를 찾아 삽입함으로써 정렬을 완성하는 알고리즘.
  - 두번째 인덱스부터 시작하여 그보다 앞쪽(왼쪽)의 인덱스와 비교하여 삽입할 위치를 지정한 후 해당 자리에 자료를 삽입하여 정렬하는 알고리즘
  - 두번째 인덱스는 첫번째 인덱스, 세번째 인덱스는 첫번째와 두번째 인덱스, 네번째 인덱스는 첫번째 두번째 세번째 인덱스와 비교한 후 자료를 삽입할 위치를 찾는다. 그 위치에 자료를 삽입하기 위해 나머지 자료를 한칸씩 뒤로 이동시킨다.
- 예시

초기상태 

8	5	6	2	4
---	---	---	---	---



오름차순  
완성상태 

2	4	5	6	8
---	---	---	---	---

## • 코드

```

void insertion_sort(int list[], int n){
    int i, j, key;

    // 인덱스 0은 이미 정렬된 것으로 볼 수 있다.
    for(i=1; i<n; i++){
        key = list[i]; // 현재 삽입될 숫자인 i번째 정수를 key 변수로 복사

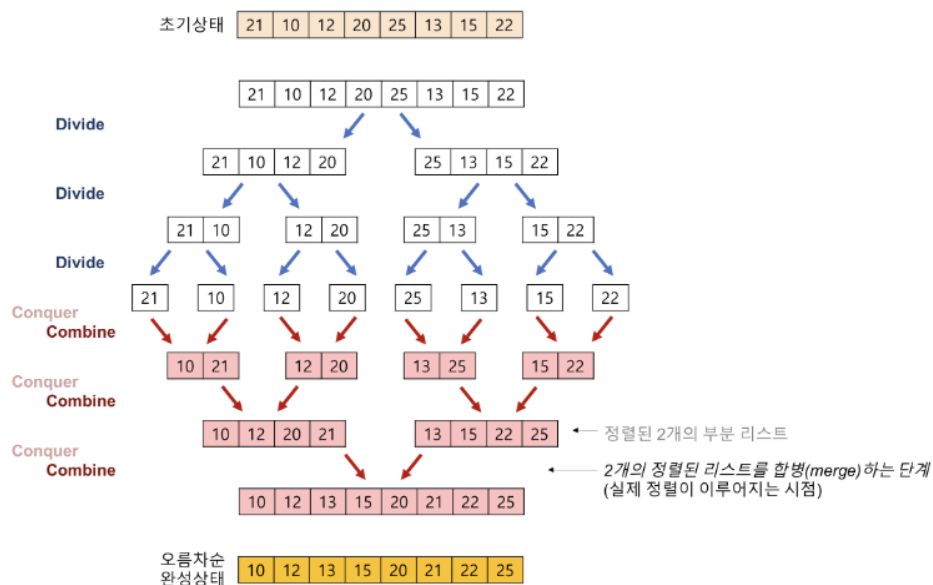
        // 현재 정렬된 배열은 i-1까지이므로 i-1번째부터 역순으로 조사한다.
        // j 값은 음수가 아니어야 되고
        // key 값보다 정렬된 배열에 있는 값이 크면 j번째를 j+1번째로 이동
        for(j=i-1; j>=0 && list[j]>key; j--){
            list[j+1] = list[j]; // 레코드의 오른쪽으로 이동
        }

        list[j+1] = key;
    }
}

```

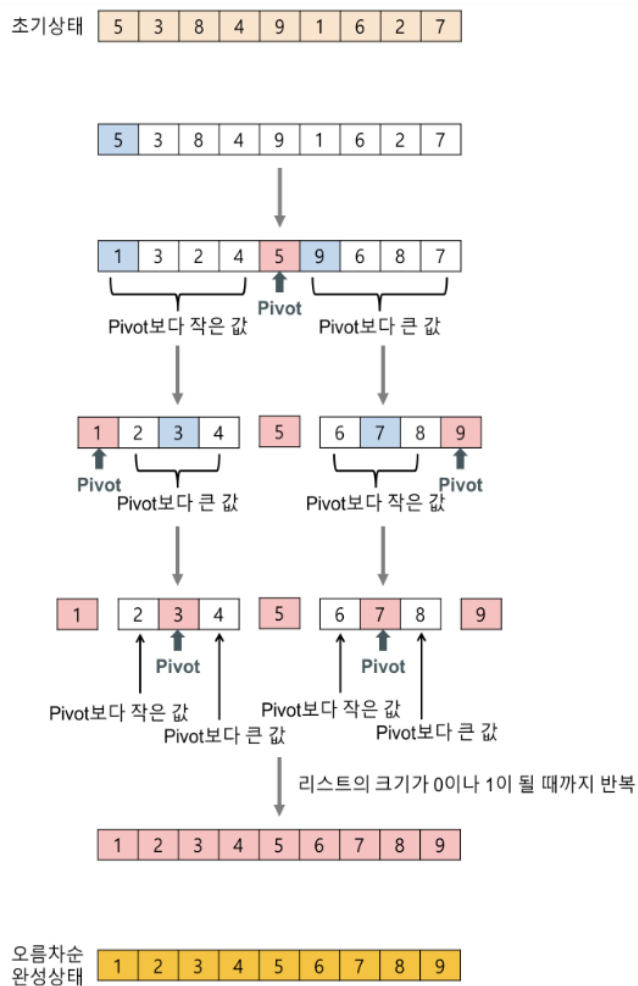
## 4) 병합 정렬

- 하나의 리스트를 두개의 균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음(분할), 부분 배열의 크기가 충분히 작지 않으면 순환 호출을 이용하여 다시 분할 정복방법을 적용하고(정복), 정렬된 부분 배열들을 하나의 배열에 합한다(결합)
- 예시



## 5) 퀵 정렬

- 분할 정복 알고리즘의 하나로, 평균적으로 매우 빠른 수행 속도를 자랑하는 정렬방법
  - 1) 리스트 안에 있는 한 요소를 선택한다. → 피벗(pivot) 선택
  - 2) 피벗을 중심으로 왼쪽 : 피벗보다 작은 요소  
 피벗을 중심으로 오른쪽 : 피벗보다 큰 요소
  - 3) 피벗을 제외한 왼쪽 리스트와 오른쪽 리스트를 다시 정렬
    - 분할된 부분 리스트에 대하여 순환 호출을 이용하여 정렬을 반복
    - 부분 리스트에서도 다시 피벗을 정하고 피벗을 기준으로 2개의 부분 리스트로 나누는 과정 반복
  - 4) 부분 리스트들이 더이상 분할이 불가능할 때까지 반복
- 예시



## 정리

- 구현이 간단하지만 비효율적인 방법 : 버블, 선택, 삽입 정렬
- 복잡하지만 효율적인 방법 : 병합, 퀵 정렬

이름	Best	Avg	Worst
버블 정렬	$n^2$	$n^2$	$n^2$
선택 정렬	$n^2$	$n^2$	$n^2$
삽입 정렬	$n$	$n^2$	$n^2$
병합 정렬	$n \log_2 n$	$n \log_2 n$	$n \log_2 n$
퀵 정렬	$n \log_2 n$	$n \log_2 n$	$n^2$