PRESENTATION

# 24주차 주제 정렬 알고리즘

## 정렬 알고리즘 ~ 퀵 정렬 ~

by mun

# 퀵 정렬

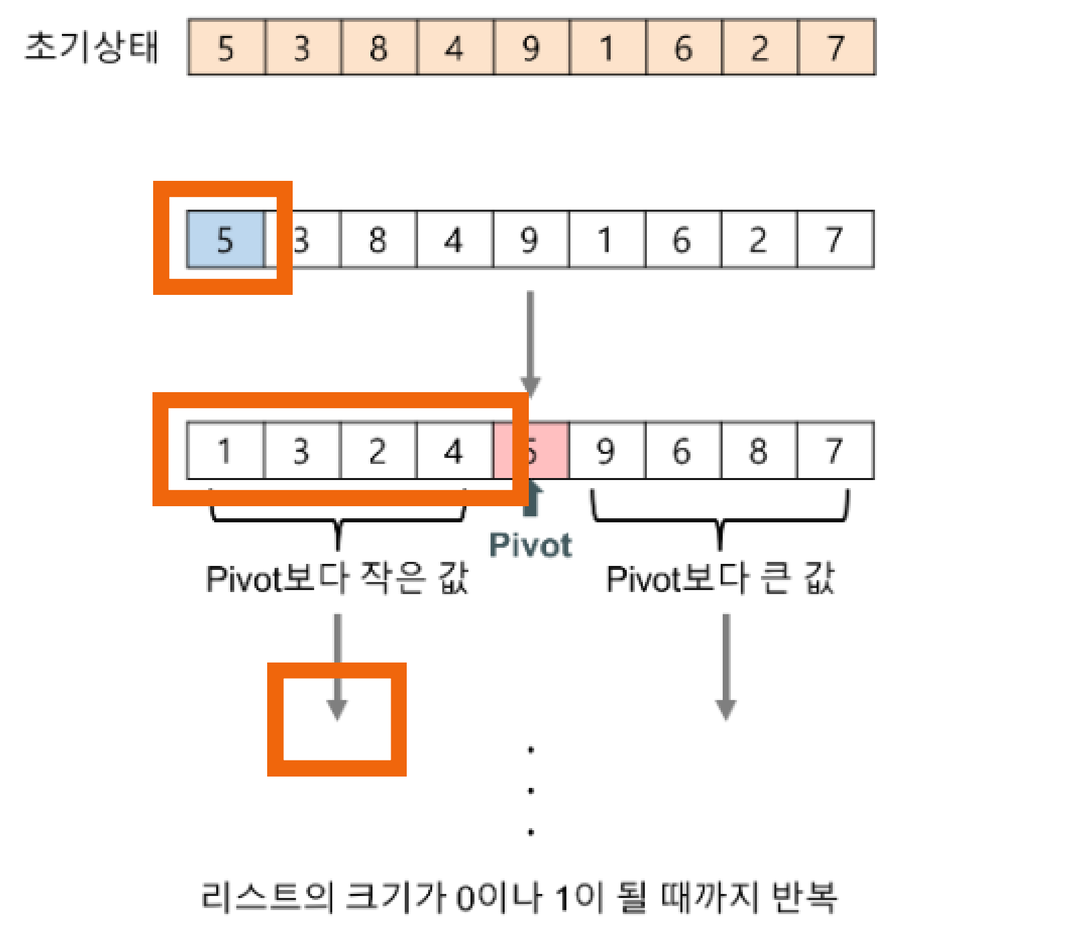
기준값을 선정해 해당 값보다
작은 데이터와 큰 데이터로 분류하는 것을 반복해 정렬하는 알고리즘

- 찰스 앤터니 리처드 호어가 개발한 범용 정렬 알고리즘
- 다른 원소와의 비교만으로 정렬을 수행하는 비교 정렬에 속한다

기준값이 어떻게 선정되는지가
시간 복잡도에 많은 영향을 미침

# 핵심 이론

**pivot을 중심으로 계속 데이터를 2개의 집합으로 나누면서 정렬하는 것**

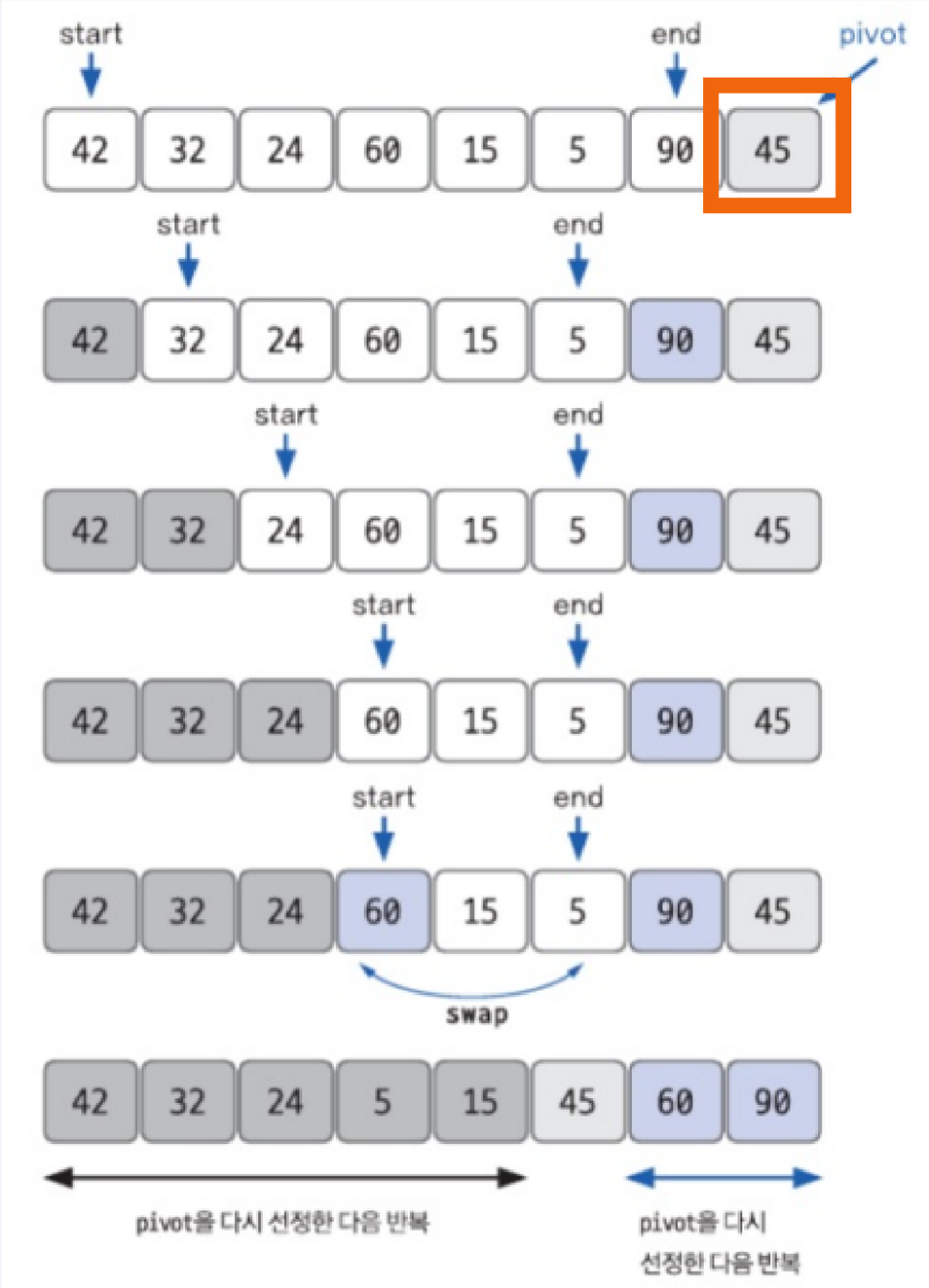



- **분할**
  입력 배열을 피벗을 기준으로 비균등하게 분할
- **정복**
  부분 배열을 정렬
  부분 배열의 크기가 작지 않다면 순환 호출을 이용해
  분할 정복 방법을 적용
- **결합**
  정렬된 부분 배열들을 하나의 배열에 합변

# 자세한 과정으로 살펴보기

어노테이션 사용한 경우

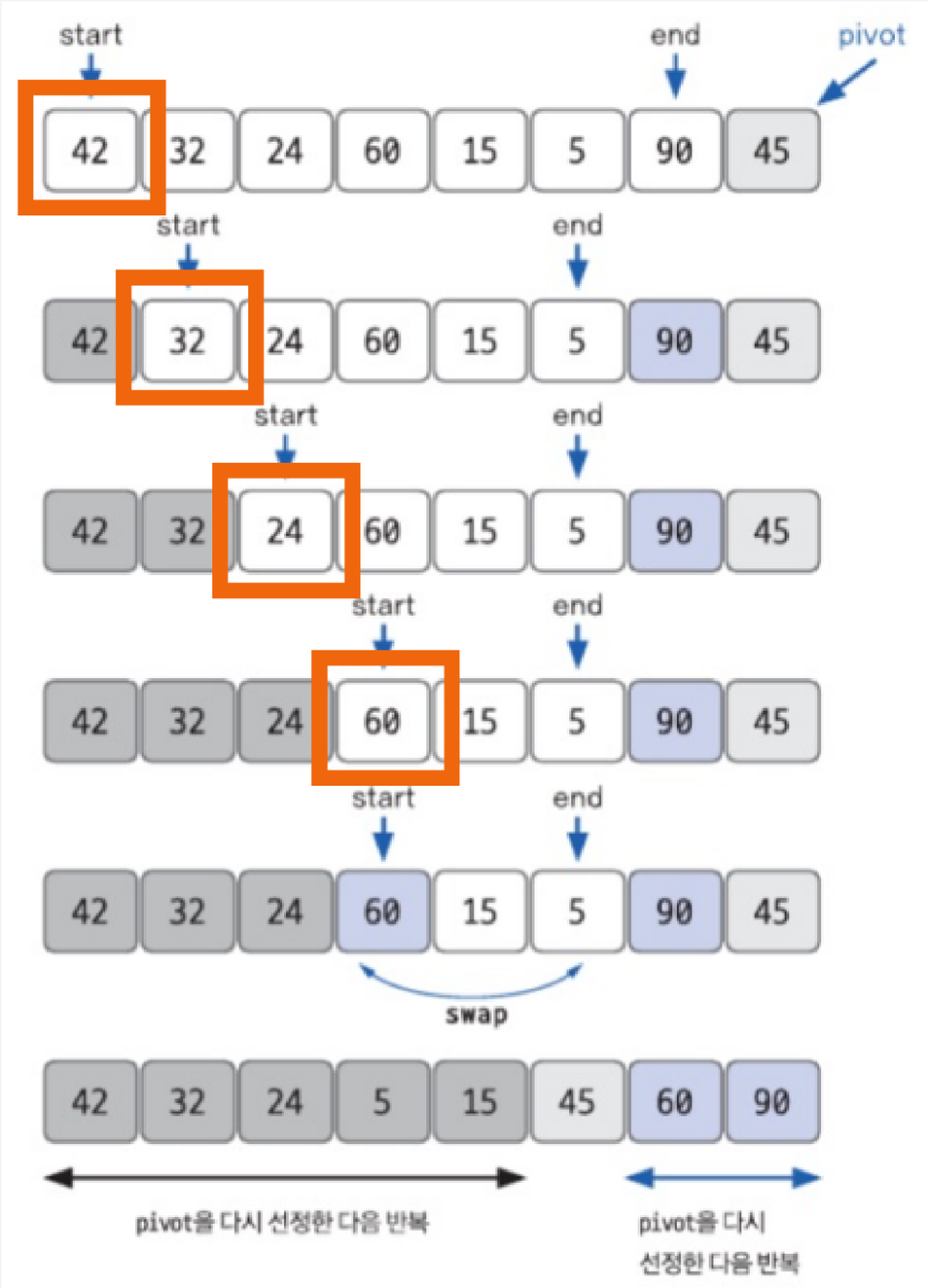
1. 데이터를 분할하는 pivot을 설정



# 자세한 과정으로 살펴보기

어노테이션 사용한 경우

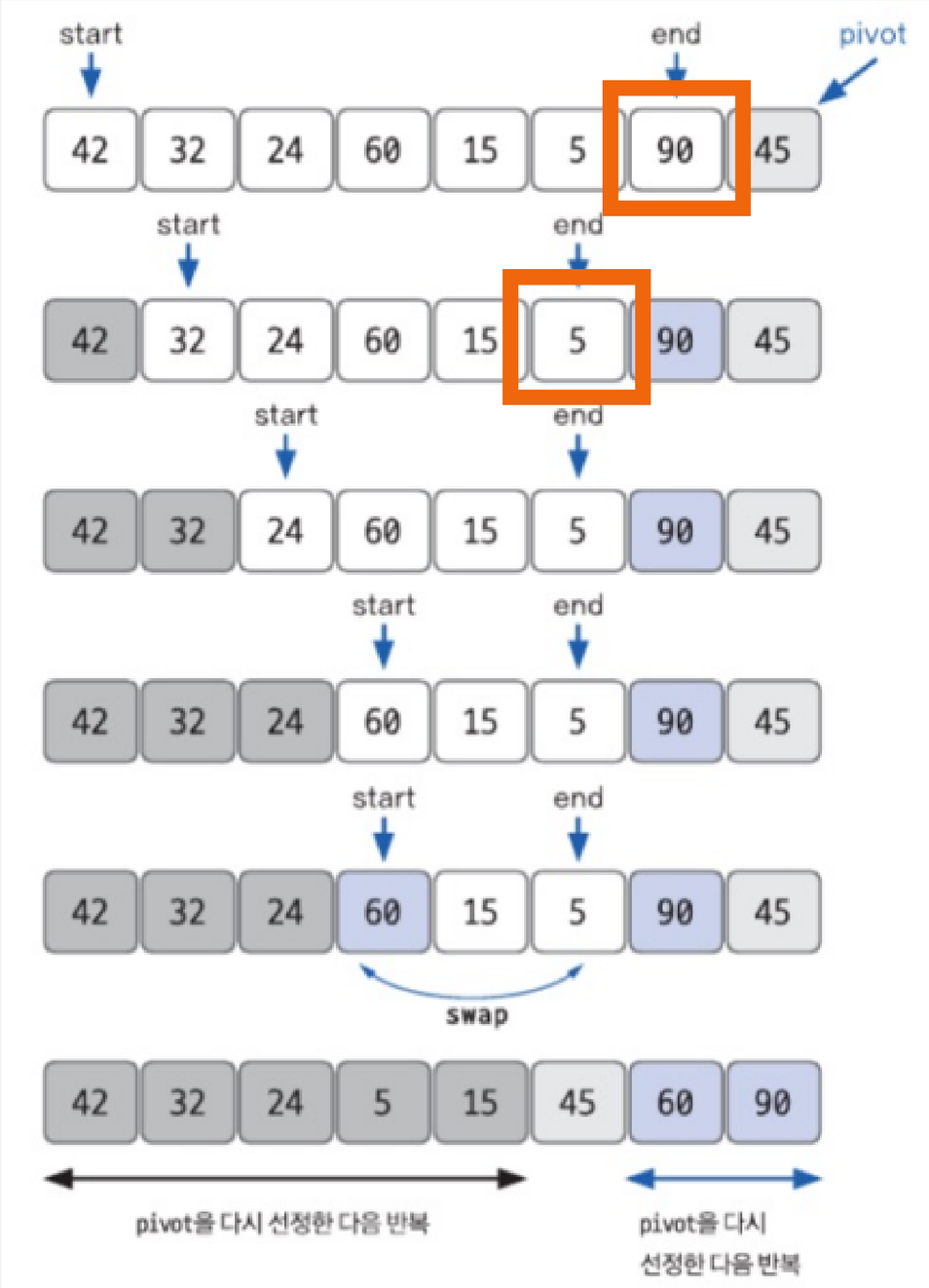
1. 데이터를 분할하는 pivot을 설정
2. pivot을 기준으로 데이터를 2개의 집합으로 분리
   a. start 데이터가 pivot보다 작으면 start를 오른쪽으로 1칸 이동



# 자세한 과정으로 살펴보기

어노테이션 사용한 경우

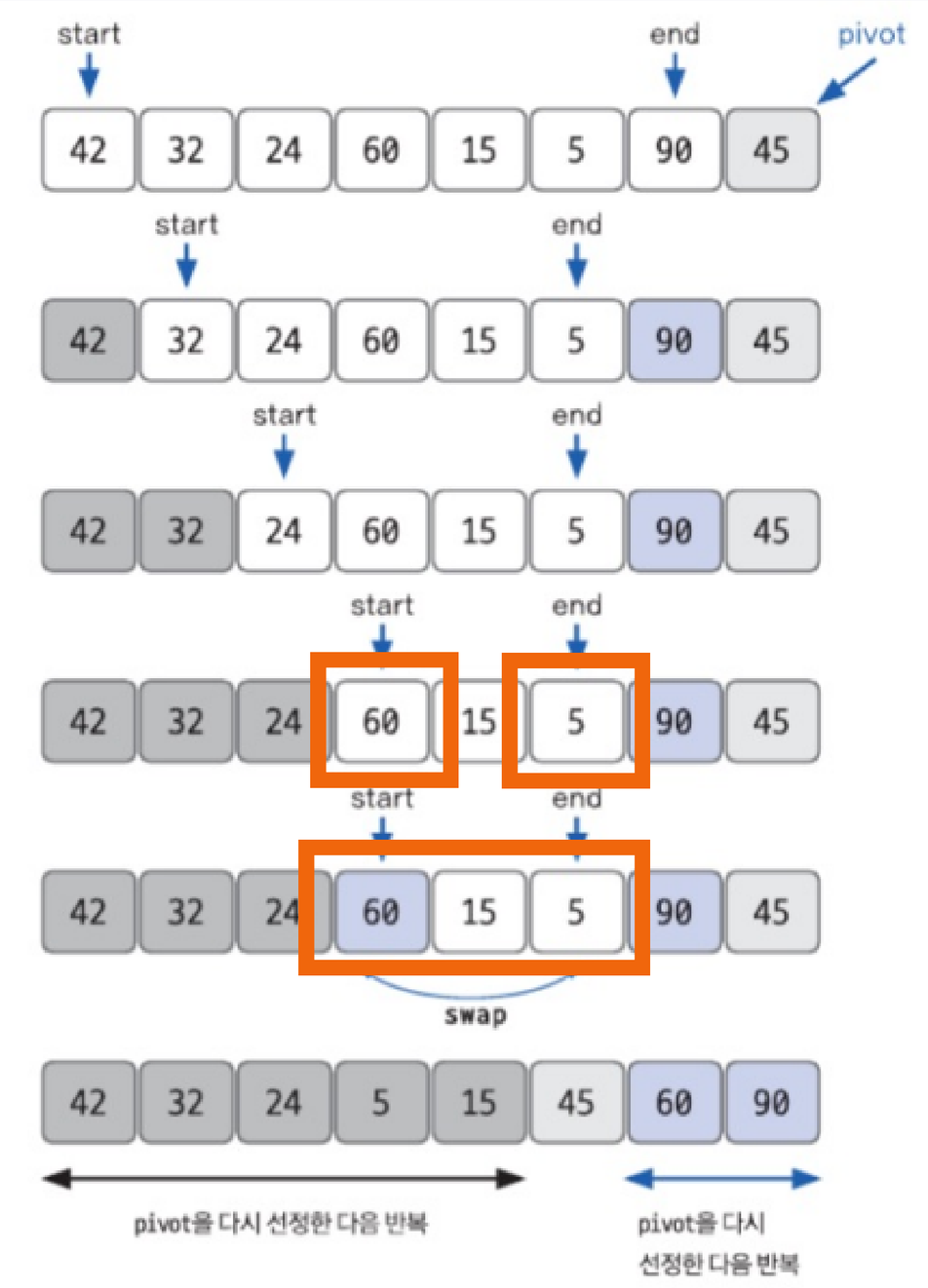
1. 데이터를 분할하는 pivot을 설정
2. pivot을 기준으로 데이터를 2개의 집합으로 분리
   a. start 데이터가 pivot보다 작으면 start를 오른쪽으로 1칸 이동
   b. end 데이터가 pivot 보다 크면 end를 왼쪽으로 1칸 이동



# 자세한 과정으로 살펴보기

**어노테이션 사용한 경우**

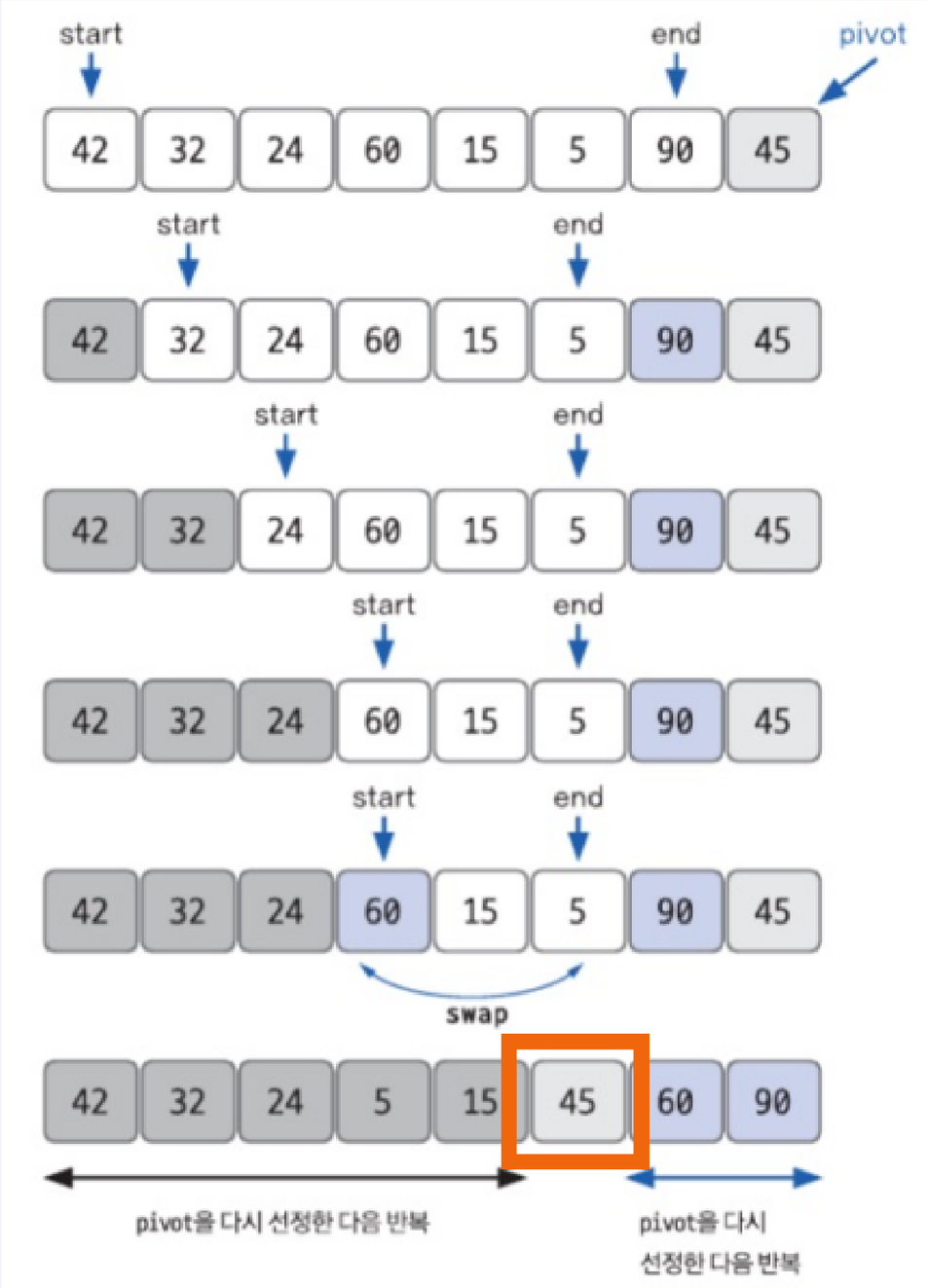
1. 데이터를 분할하는 pivot을 설정
2. pivot을 기준으로 데이터를 2개의 집합으로 분리
   a. start 데이터가 pivot보다 작으면 start를 오른쪽으로 1칸 이동
   b. end 데이터가 pivot 보다 크면 end를 왼쪽으로 1칸 이동
   c. start 데이터가 pivot보다 크고 end데이터가 pivot보다 작으면 start, end 데이터를 swap,
      start는 오른쪽, end는 왼쪽으로 1칸씩 이동
   d. start와 end가 만날때까지 위 과정을 반복



# 자세한 과정으로 살펴보기

어노테이션 사용한 경우

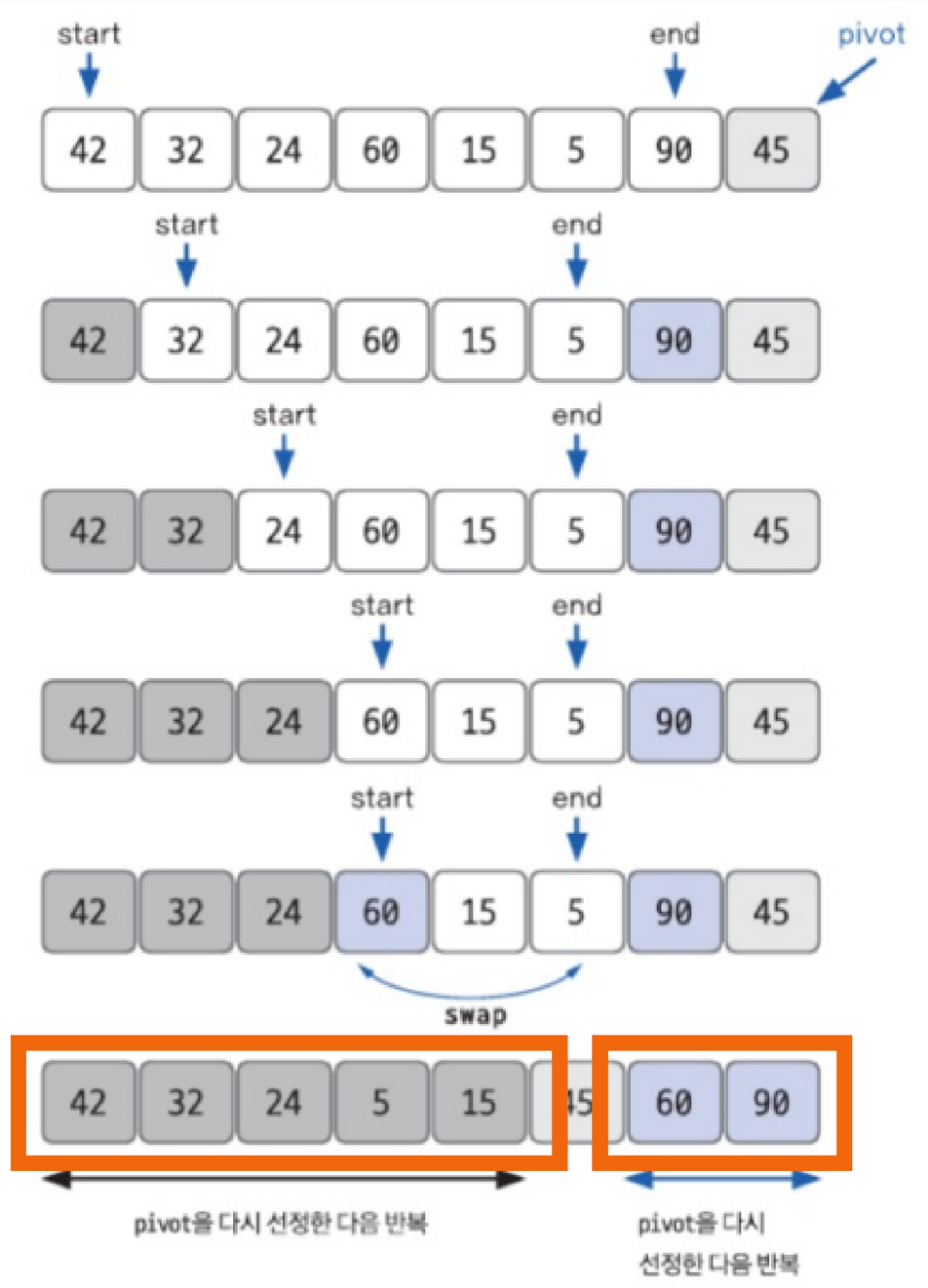
1. 데이터를 분할하는 pivot을 설정
2. pivot을 기준으로 데이터를 2개의 집합으로 분리
   a. start 데이터가 pivot보다 작으면 start를 오른쪽으로 1칸 이동
   b. end 데이터가 pivot 보다 크면 end를 왼쪽으로 1칸 이동
   c. start 데이터가 pivot보다 크고 end데이터가 pivot보다 작으면 start, end 데이터를 swap,
      start는 오른쪽, end는 왼쪽으로 1칸씩 이동
   d. start와 end가 만날때까지 위 과정을 반복
   e. start와 end가 만나면 만난 지점의 데이터와 pivoit을 비교,
      pivot 데이터가 크면 만난 지점의 오른쪽에,
      작으면 만난 지점의 왼쪽에 pivot 데이터를 삽입



# 자세한 과정으로 살펴보기

**어노테이션 사용한 경우**

1. 데이터를 분할하는 pivot을 설정
2. pivot을 기준으로 데이터를 2개의 집합으로 분리
   a. start 데이터가 pivot보다 작으면 start를 오른쪽으로 1칸 이동
   b. end 데이터가 pivot 보다 크면 end를 왼쪽으로 1칸 이동
   c. start 데이터가 pivot보다 크고 end데이터가 pivot보다 작으면 start, end 데이터를 swap,
      start는 오른쪽, end는 왼쪽으로 1칸씩 이동
   d. start와 end가 만날때까지 위 과정을 반복
   e. start와 end가 만나면 만난 지점의 데이터와 pivoit을 비교,
      pivot 데이터가 크면 만난 지점의 오른쪽에,
      작으면 만난 지점의 왼쪽에 pivot 데이터를 삽입
3. 분리 집합에서 다시 pivot 을 선정
4. 분리 집합이 1개 이하가 될 때까지 위 과정을 반복



# 장단점

| 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 |

**장점**

- 속도가 빠르다
- 추가 메모리 공간을 필요로 하지 않는다

**단점**

- 정렬된 리스트에 대해서는 오히려 수행 시간이 더 많이 걸린다

# 구현해보기

```java
public void quickSort(int[] arr, int left, int right) {
    // base condition
    if (left >= right) {
        return;
    }
    int pivot = arr[right];

    int sortedIndex = left;
    for (int i = left; i < right; i++) {
        if (arr[i] <= pivot) {
            swap(arr, i, sortedIndex);
            sortedIndex++;
        }
    }
    swap(arr, sortedIndex, right);
    quickSort(arr, left, sortedIndex - 1);
    quickSort(arr, sortedIndex + 1, right);
}

private void swap(int[] arr, int i, int j) {
    int tmp = arr[i];
    arr[i] = arr[j];
    arr[j] = tmp;
}
```

재귀 형태로 구현 가능

# 구현해보기

**11004 K번째 수**

## K번째 수

☆

| 시간 제한 | 메모리 제한 | 제출 | 정답 | 맞힌 사람 | 정답 비율 |
|---|---|---|---|---|---|
| 2 초 | 512 MB | 59768 | 20818 | 14688 | 41.412% |

### 문제

수 N개 $A_1$, $A_2$, ..., $A_N$이 주어진다. A를 오름차순 정렬했을 때, 앞에서부터 K번째 있는 수를 구하는 프로그램을 작성하시오.

### 입력

첫째 줄에 N($1 \le N \le 5,000,000$)과 K ($1 \le K \le N$)이 주어진다.

둘째에는 $A_1$, $A_2$, ..., $A_N$이 주어진다. ($-10^9 \le A_i \le 10^9$)

### 출력

A를 정렬했을 때, 앞에서부터 K번째 있는 수를 출력한다.

### 예제 입력 1 복사

```
5 2
4 1 2 3 5
```

### 예제 출력 1 복사

```
2
```

# 구현해보기

```java
public static int partition(int[] array, int left, int right) {
    int mid = (left + right) / 2;
    swap(array, left, mid);
    int pivot = array[left];
    int i = left, j = right;
    while (i < j) {
        while (pivot < array[j]) j--;
        while (i < j && pivot >= array[i]) i++;
        swap(array, i, j);
    }
    array[left] = array[i];
    array[i] = pivot;
    return i;
}

public static void swap(int[] array, int a, int b) {
    int temp = array[b];
    array[b] = array[a];
    array[a] = temp;
}
```

앞, 뒤 포인터 이동

```java
public static void quicksort(int[] array, int left, int right) {
    if (left >= right) {
        return;
    }
    int pi = partition(array, left, right);
    if(pi+1 == k) return;
    else if(pi+1<k)
      quicksort(array, pi + 1, right);
    else
      quicksort(array, left, pi - 1);

}
```

분리 집합끼리 퀵소트

# 구현해보기

```java
public static int partition(int[] array, int left, int right) {
    int mid = (left + right) / 2;
    swap(array, left, mid);
    int pivot = array[left];
    int i = left, j = right;
    while (i < j) {
        while (pivot < array[j]) j--;
        while (i < j && pivot >= array[i]) i++;
        swap(array, i, j);
    }
    array[left] = array[i];
    array[i] = pivot;
    return i;
}

public static void swap(int[] array, int a, int b) {
    int temp = array[b];
    array[b] = array[a];
    array[a] = temp;
}
```

앞, 뒤 포인터 이동

| 제출 번호 | 아이디 | 문제 | 결과 | 메모리 | 시간 |
|---|---|---|---|---|---|
| 76778952 | dalmun | 11004 | 맞았습니다!! | 352992 KB | 4892 ms |

```java
public static void quicksort(int[] array, int left, int right) {
    int pi = partition(array, left, right);
    if(pi+1 == k) return;
    else if(pi+1<k)
      quicksort(array, pi + 1, right);
    else
      quicksort(array, left, pi - 1);

}
```

분리 집합끼리 퀵소트

# 문제 유형에 따라
# 알맞은 정렬 알고리즘을 선택해봐요

# 감사합니다

참고 : Do it! 알고리즘 코딩 테스트 : 자바 편 (저 김종관), https://ko.wikipedia.org/wiki/퀵정렬

https://gmlwjd9405.github.io/2018/05/10/algorithm-quick-sort.html