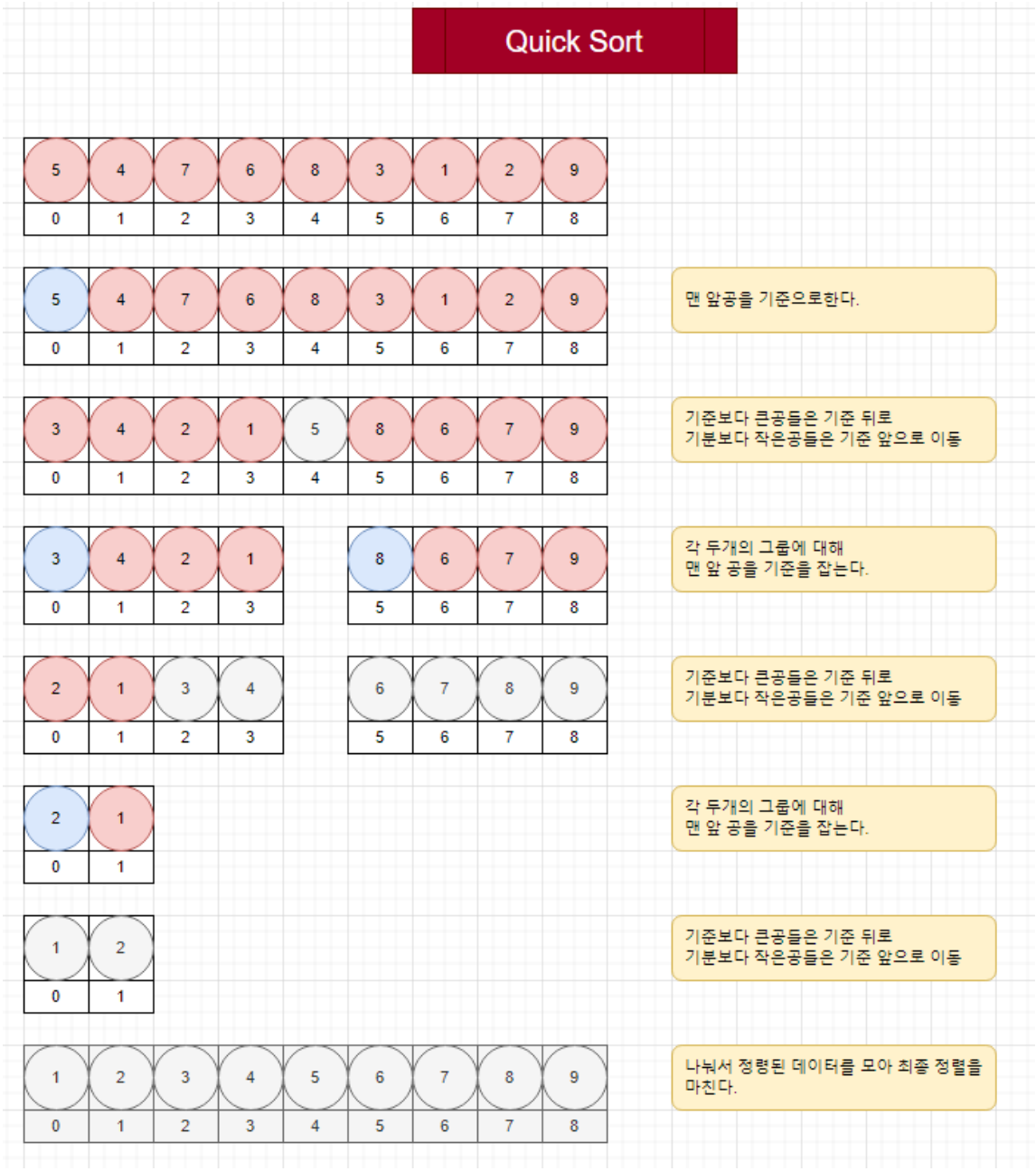
Insertion Sort

Date	
작성일시	@2022년 8월 3일 오전 10:29
강의 번호	
유형	
강사명	
강의자료	☐
노션 복습	☐
코딩 복습	☐
주말숙제(교제)	☐
정리	☐

▼ 퀵 정렬_Quick Sort

- 데이터를 대소그룹 둘로 나누어 분해한 후에 전체를 최종적으로 정렬하는 알고리즘이다.
- Divide and conquer(분할 정복법)
- 퀵정렬은 대량의 데이터를 정렬할 때 매우 자주 사용된다.
- 유명한 알고리즘 중에서도 실제로 많이 사용되는 빈도가 가장 높고 중요한 알고리즘이기도 하다.
- 퀵정렬은 '기준값을 선택한 후 그 보다 작은 데이터 그룹과 큰데이터 그룹으로 나눈다.' 라는 처리를 반복수행하여 데이터를 정렬하게 된다.

▼ Process



▼ Algorithm

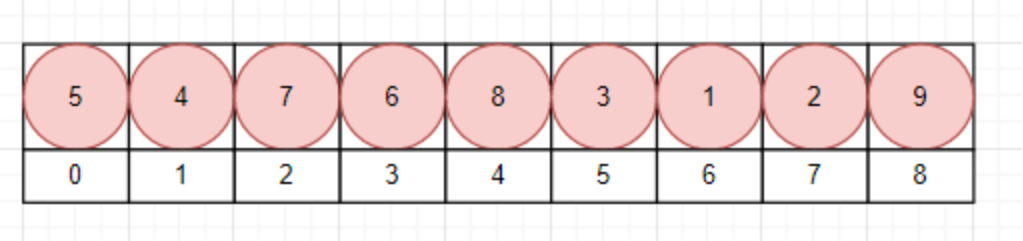
퀵 정렬은 크게 2개의 처리로 구성된다.

▼ (1) 기준값을 경계로 데이터를 대소로 나누는 처리

- 퀵정렬의 핵심은 데이터를 대소로 나누는 처리이다.
- 베열의 왼쪽과 오른쪽부터 각각 변수를 움직여 대소로 정렬하자.

> 👉 기준값보다 작은 공을 기준값의 앞으로 이동시키고 기준값보다 큰 공은 뒤로 이동시키는 것이 바로 퀵정렬의 초석이 되는 처리이다.

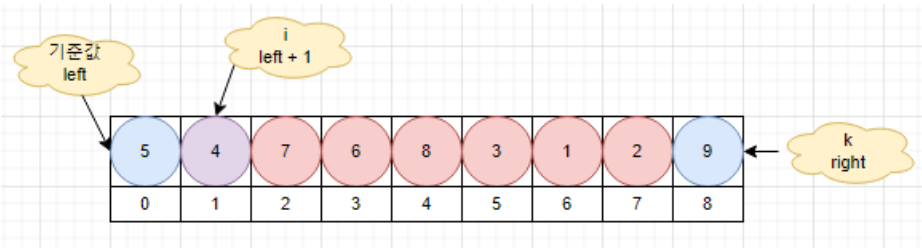
1. 배열설정 : 먼저 배열을 준비하자, 정수형 배열로 이름은 arr 요소의 수는 9개로 정한다.



⇒ 변수설정_변수는 5개

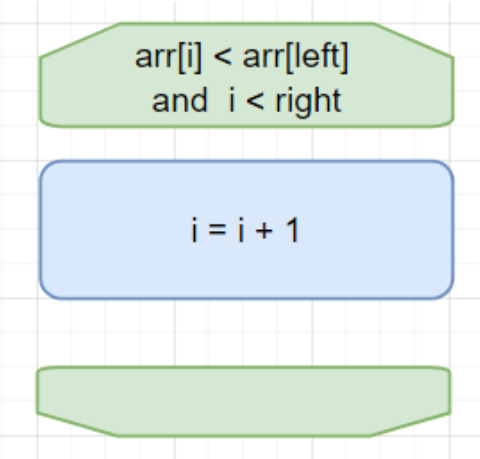
1. left - 정렬 범위에서 맨 앞 요소에 첨자를 넣는 변수
2. right - 정렬 범위에서 맨 끝 요소에 첨자를 넣는 변수
3. i - 기준값보다 큰 요소를 찾기 위한 변수
4. k - 기준값보다 작은 요소를 찾기 위한 변수
5. w - 데이터 교환용 임시 변수 temp

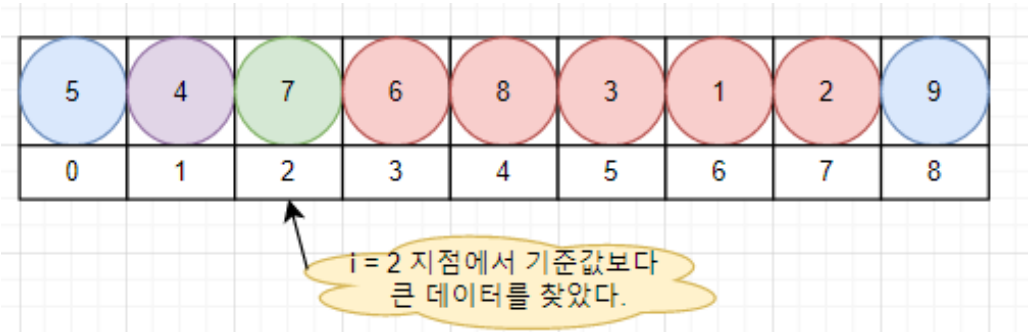
> 👉 이 5개의 변수를 사용하여 우선 left와 right에 각각 정렬 범위 맨 앞 요소의 첨자와 마지막 요소의 첨자를 대입한다. 따라서 이번에는 (처음에는) left 0, right 8이 된다. 기준은 맨 앞 요소로 하기 때문에 arr[left]가 된다. 그리고 i에 left의 하나 오른쪽에 left +1 로 정하고 k에는 right을 대입한다.



2. 변수 i를 사용하여 기준값보다 큰 요소 찾기

👉 i는 '기준갑보다 큰 요소를 찾는 변수'이다. 현재 위치에서 하나씩 오른쪽으로 이동하면서 기준값보다 큰 요소가 있는지 확인하고 발견되면 그곳에서 멈춘다.
⇒ arr[i] > arr[left]
*2가지 조건
1. arr[left]보다 큰 값을 찾고
2. 오른쪽 끝까지 반복.



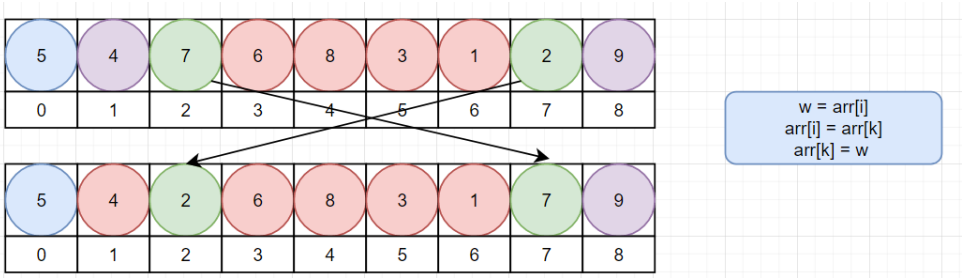


👉 기준값보다 큰 요소를 발견했기 때문에 i는 일단 여기서 멈춘다. 그대로 반대쪽 변수 k 즉 작은 값 찾기로 넘어간다.

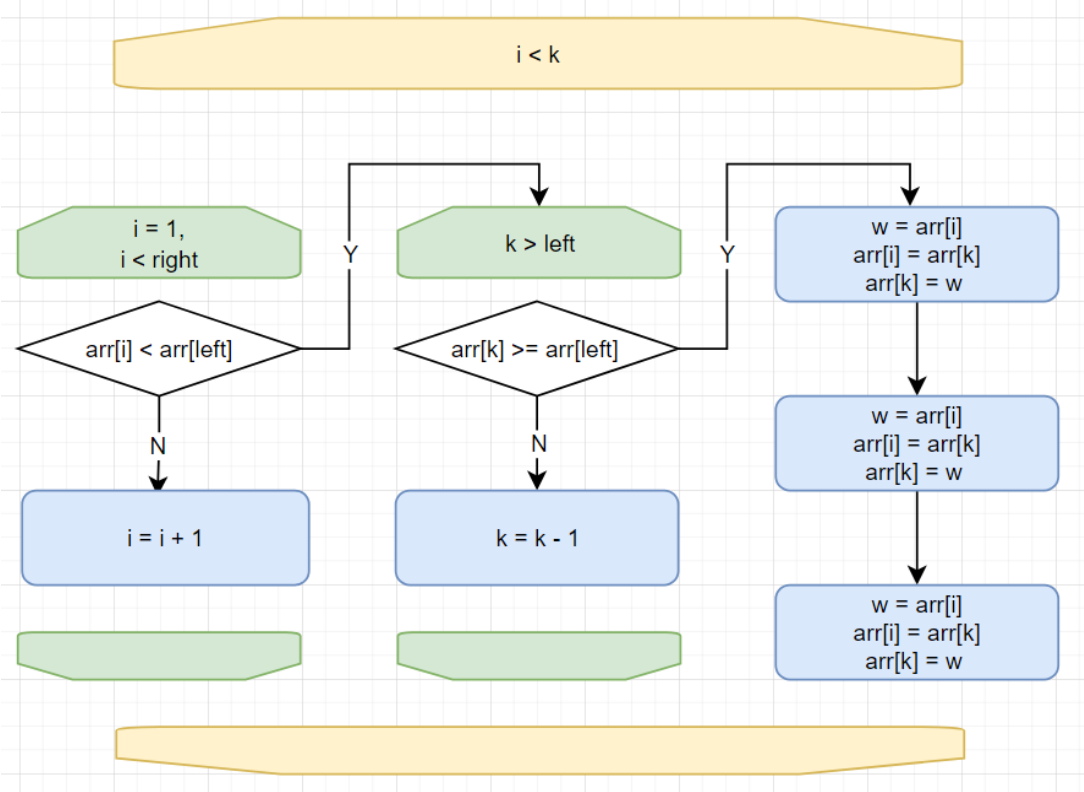
3. 변수 k를 사용하여 기준값보다 작은 요소 찾기

👉 기준값보다 작은 요소를 발견했기 때문에 k도 일단 여기서 멈춘다.

▼ 두 데이터를 교환



▼ (2) 나눈 데이터에 대해 반복적으로 똑같은 작업을 실행



▼ Java code

```
package QuickSort;

import java.util.*;

public class QuickSort {

  public static void main(String[] args) {
    int[] arr = {5,4,7,6,8,3,1,2,9};
    System.out.println(Arrays.toString(arr)); //배열 출력

    arr = quickSort(arr,0,arr.length-1);//어레이, 시작, 끝
    System.out.println(Arrays.toString(arr));
  }

  static int[] quickSort(int[] arr, int start, int end) {
    int p = partition(arr, start, end);

    if(start < p-1) {
      quickSort(arr, start, p-1);
    }else if(p < end) {
      quickSort(arr, p, end);
    }
    return arr;
  }
```

```
  static int partition(int[] arr, int start, int end) {
    int pivot = arr[(start+end)/2]; //기준값을 가운데로
    while(start <= end) {
      while(arr[start] < pivot) {
        start++;
      }
      while(arr[end]>pivot) {
        end--;
      }
      if (start <= end) {
        int tmp = arr[start];
        arr[start] = arr[end];
        arr[end] = tmp;
        start++;
        end--;
      }
    }
    return start;
  }
 }
Outputs :
[5, 4, 7, 6, 8, 3, 1, 2, 9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```