Sort_Quick

작성일시	@2022년 8월 3일 오전 9:53
강의 번호	
유형	
강사	
자료	
복습	☐
날짜	@2022년 8월 2일

퀵 정렬 Quick Sort

데이터를 대소그룹 둘로 나누어 분해한 후에 전체를 최종적으로 정렬하는 방식의 알고리즘이다.

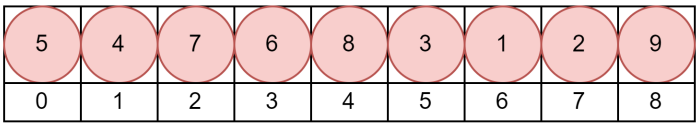
Divide and conquer (분할 정복법)

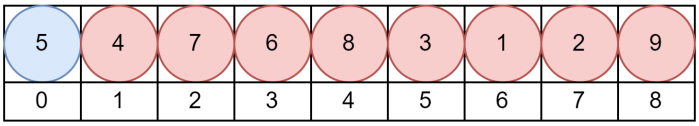
퀵 정렬은 대량의 데이터를 정렬할 때 매우 자주 사용된다. 유명한 알고리즘 중에서도 실제로 많이 사용되는 빈도가 가장높고 중요한 알고리즘이기도 하다.

퀵 정렬 '기준값을 선택한 후 그보다 작은 데이터 그룹과 큰 데이터 그룹으로 나누다.' 라는 처리를 반복수행하여 데이터를 정렬하게 된다.

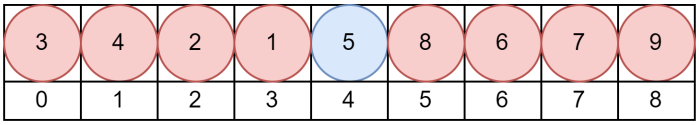
Algorithm

Quick Sort

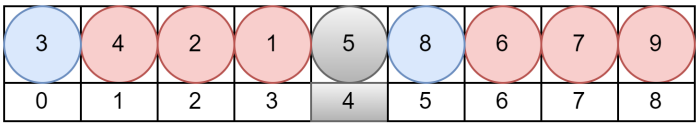


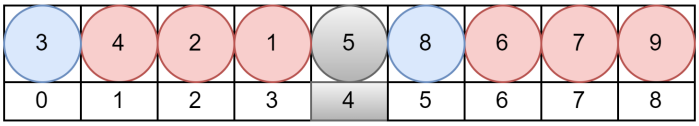


맨 앞의 공을 기준으로 한다.

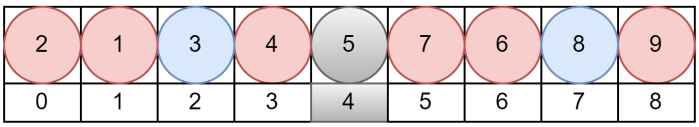


기준보다 큰 공은 기준 뒤로 작은 공들은 앞으로 이동

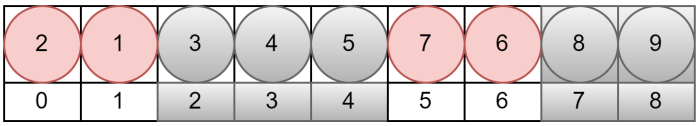


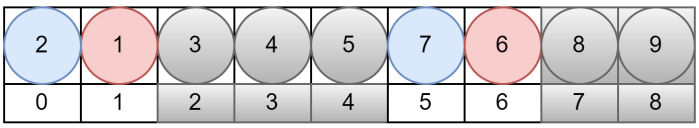


맨 앞의 공을 기준으로 한다.

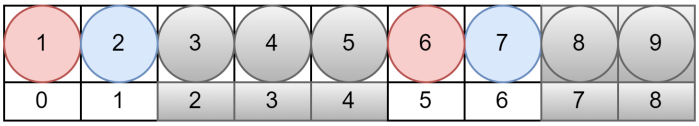


기준보다 큰 공은 기준 뒤로 작은 공들은 앞으로 이동

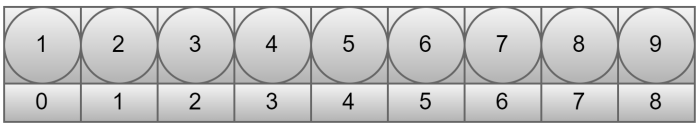




맨 앞의 공을 기준으로 한다.



기준보다 큰 공은 기준 뒤로 작은 공들은 앞으로 이동



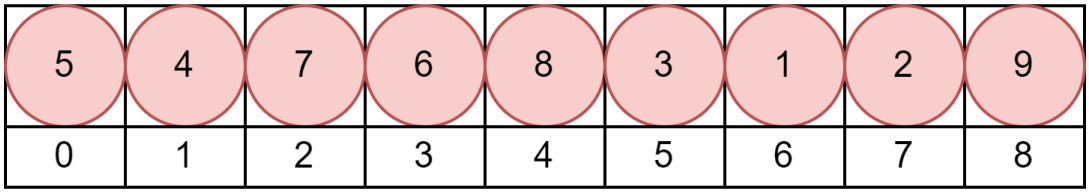
퀵 정렬의 Algorithm & Flow Chart

크게 2가지의 처리로 구성된다.

1. **기준값을 경계로 데이터를 대소로 나누는 처리**
   - 데이터를 대소로 나눈는 처리 이다.
   - 배열의 왼쪽과 오른쪽부터 각각 변수를 움직여 대소로 정렬

기준값보다 작은공은 앞으로 이동시키고 기준값보다 큰 공을 뒤로 이동시키는 것이 바로 퀵정렬의 초석이다.

배열설정 (정수형, 이름은 arr, 요소수는 9개, 첨자는 0~8)



변수설정

변수는 5개를 준비한다.

left - 정렬범위에서 맨앞 요소에 첨자를 넣는 변수

right - 정렬범위에서 맨 끝 요소에 첨자를 넣는 변수

i - 기준값보다 큰 요소를 찾기위한 변수

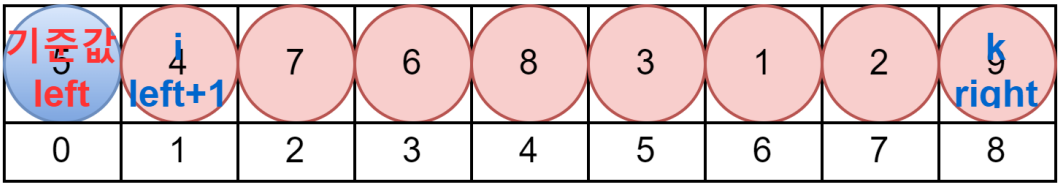
k - 기준값보다 작은 요소를 찾기위한 변수

w - 데이터 교환용 임시 변수(=temp)

5개의 변수를 사용하여, 우선 left와 right의 맨앞 요소와 마지막 요소의 첨자를 대입한다.

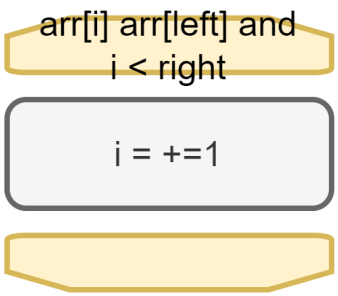
left 0 right 8,

기준은 맨앞요소 이기에 arr[left], i에 left+1, k는 right을 대입한다.

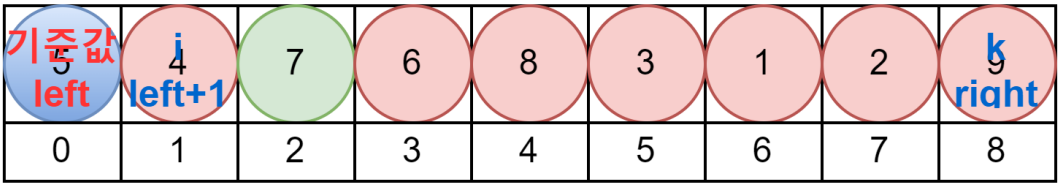


변수 i를 사용하여 기준 값보다 큰 요소를 찾는다. 현재위치에서 하나씩 오른쪽으로 이동하면서 기준값보다 큰 요소가 있는지 확인하고 발견되면 그곳에서 멈춘다.

arr[i] > arr[left]

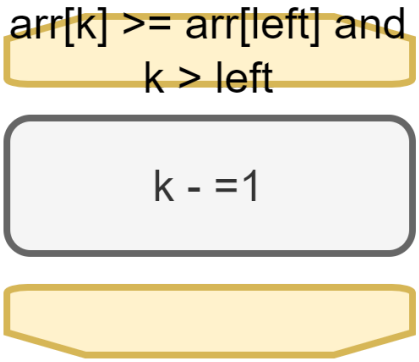


2가지 조건 만족 할때까지 반복

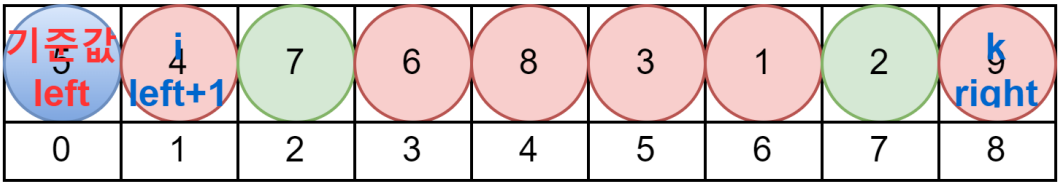


기준 값보다 큰요소를 발견 했기 때문에 i는 일단 여기서 멈춘후 반대쪽 변수k, 즉 작은 값을 찾는다.

변수 k를 사용하여 기준값보다 작은값을 찾는다. 현재 k위치에서 하나씩 왼쪽으로 이동하면서 기준값보다 작은 요소가 있는지 확인하고 발견되고 그곳에서 멈춘다.

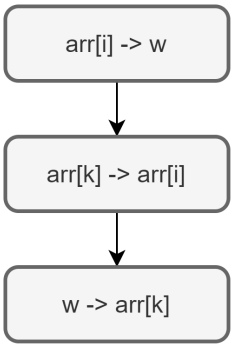


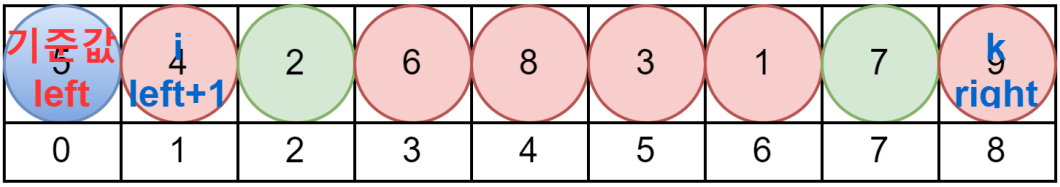
2가지 조건 만족할때까지 반복



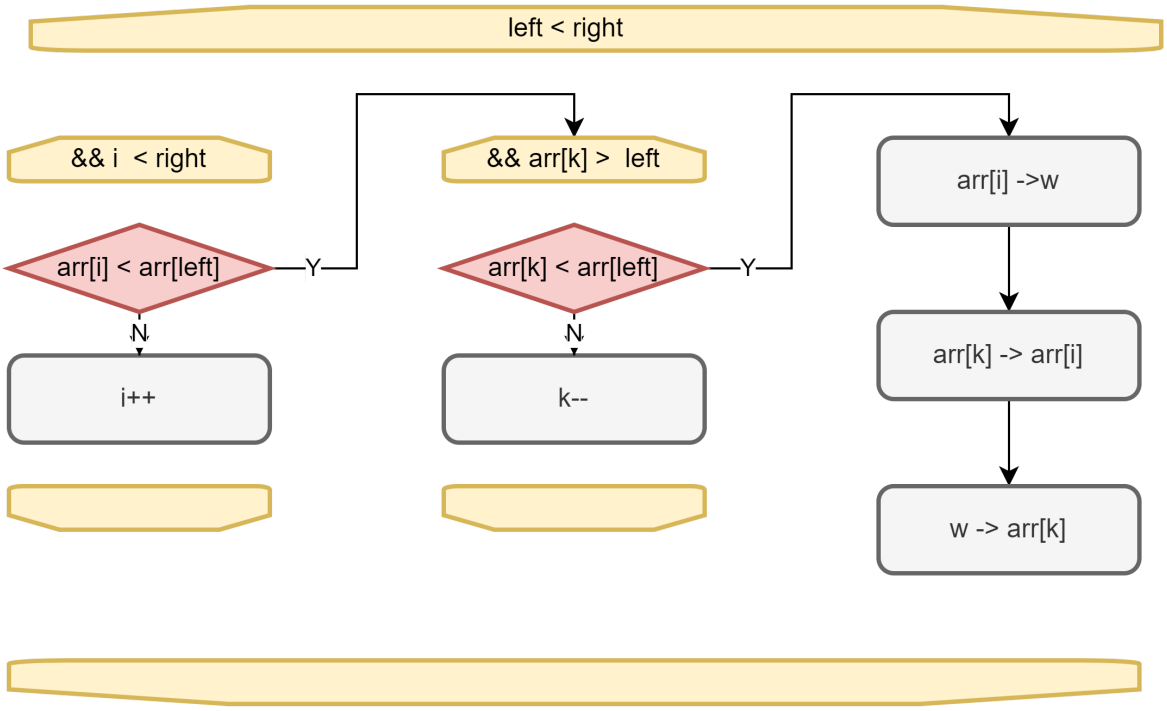
기준값보다 작은 요소를 발견 했기때문에 k도 여기에서 멈춘다.

그 이후 큰 데이터와 작은데이터를 교환한다.





2. **나눈 데이터에 대해 반복적으로 똑같은 작업을 실행**



퀵 정렬의 코드 (플로우차트와는 살짝 다름.)

```
package quicksort;

import java.util.Arrays;

public class QuickSort {

  public static void main(String[] args) {
    int[] arr = {5,4,7,6,8,3,1,2,9};
    arr = quickSort(arr, 0, arr.length-1);
    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);}
    if(p<end) {
      quickSort(arr, p, end);}
    return arr;
  }

  static int partition(int[] arr, int start, int end) {
    int pivot = arr[(start+end)/2];
    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;
  }

}
```