24주차 발표_민서연

# 정렬

# 01 정렬

## 개념

목록 안에 저장된 요소들을 특정한 순서대로 재배치하는 알고리즘

# 01 정렬

## 특징

시간 복잡도

안정성 정렬을 했을 때 중복된 값들의 순서가 변하지 않는 것을 말함

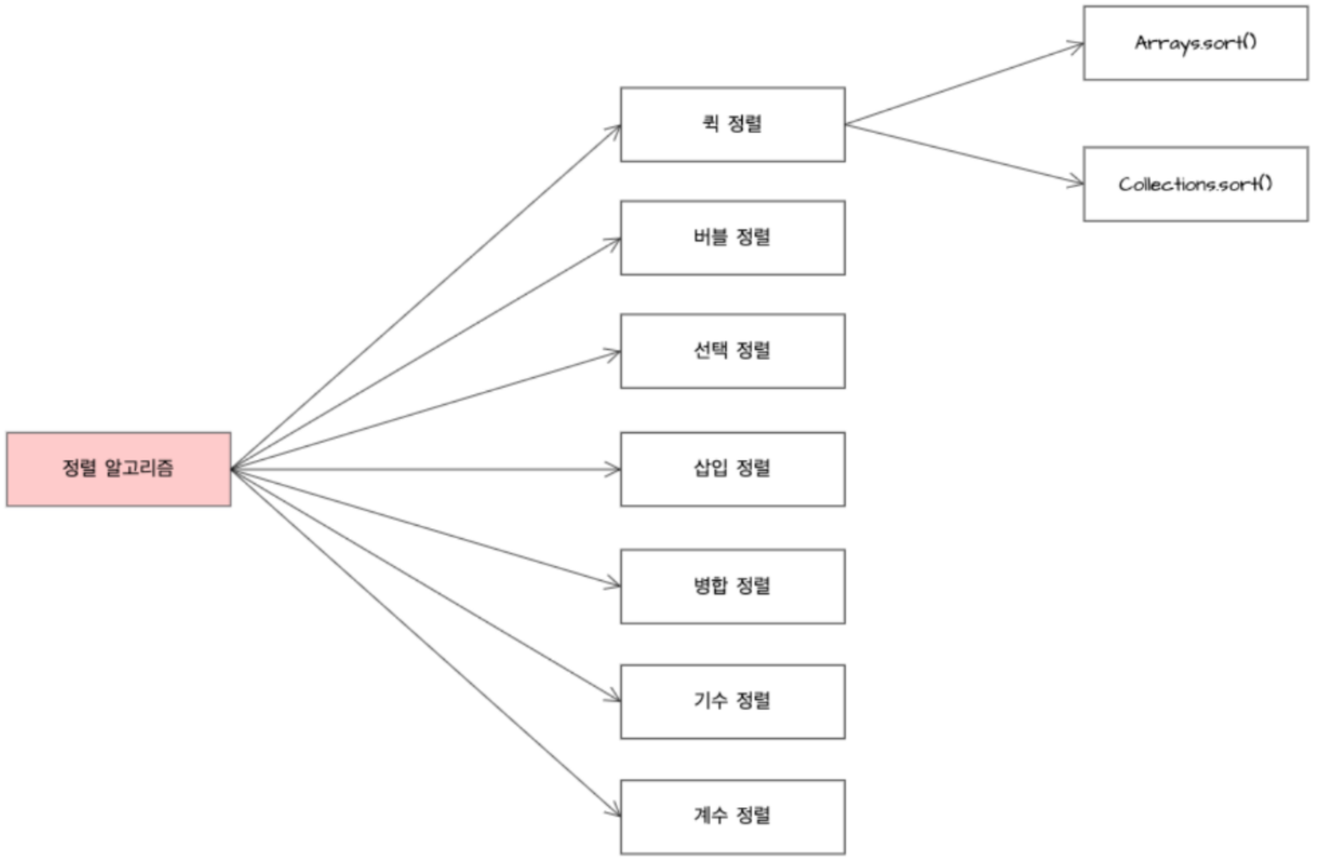
예)
arr = [1, 2, 5, 8(1), 3, 9, 8(2)]
arr = [1, 2, 3, 5, 8(1), 8(2), 9] (stable)
arr = [1, 2, 3, 5, 8(2), 8(1), 9] (unstable)

추가 메모리 사용 몇몇 정렬 알고리즘은 추가적인 메모리 공간을 필요로 해서 선택할 때 고려해야함

알고리즘의 복잡성 알고리즘의 이해와 구현의 어려움

# 01 정렬

## 종류


정렬 알고리즘
퀵 정렬
버블 정렬
선택 정렬
삽입 정렬
병합 정렬
기수 정렬
계수 정렬
Arrays.sort()
Collections.sort()

# 02 버블정렬

## Bubble Sort

인접한 두 원소를 비교해서 큰 값을 오른쪽으로 이동시키는 방식으로 정렬하는 알고리즘이다.

해당 정렬 방식은 정렬할 원소의 개수가 적을 때나 정렬할 원소의 개수가 많더라도 이미 거의 정렬 된 상태 일 때 사용될 수 있다.

시간복잡도가 느려서 간단한 정렬에 유용하다.

# 02 버블정렬


5 3 8 1 2 7
초기 상태
5와 3을 교환
교환 없음
8과 1을 교환
8과 2를 교환
8과 7을 교환
가장 큰 값 정렬


초기 상태
첫 번째 큰 값
두 번째 큰 값
세 번째 큰 값
네 번째 큰 값
다섯 번째 큰 값
정렬 완료

# 02 버블정렬

```java
public class BubbleSort {
    1 usage
    public static void bubbleSort(int[] array){
        for(int i = 0; i < array.length; i++){
            for(int j = 0; j < array.length - i - 1; j++){
                if(array[j] > array[j + 1]){
                    swap(array, j, target: j + 1);
                }
            }
        }
    }
    1 usage
    public static void swap(int[] array, int source, int target){
        int temp = array[source];
        array[source] = array[target];
        array[target] = temp;
    }
    2 usages
    public static void printArray(int[] array){
        for(int data : array){
            System.out.print(data + " ");
        }
        System.out.println();
    }
    public static void main(String[] args){
        int[] item = new int[] {4, 8, 1, 3, 5, 2, 7, 6};
        System.out.println("정렬 전");
        printArray(item);
        bubbleSort(item);
        System.out.println("정렬 후");
        printArray(item);
    }
}
```

끝을 제외한 나머지 길이 만큼 반복

```
/Library/Java/JavaVirtualMachines/j
정렬 전
4 8 1 3 5 2 7 6
정렬 후
1 2 3 4 5 6 7 8
```

# 02 버블정렬

## 장점

- 구현이 매우 간단하고 소스가 직관적이다.
- 정렬하고자 하는 배열 안에서 교환하는 방식으로 In-place 정렬이다.
- stable 정렬이다.

## 단점

- 시간 복잡도가 최악, 최선, 평균 모두 O(n^2)으로 비효율적이다.
- 정렬된 상태에서 새로운 데이터가 추가되면 정렬 효율이 좋지 않다.

# 03 정렬과 페이징 처리

## JPA Query method

스프링 데이터 jpa는 쿼리 메서드 기능을 제공하기 때문에
메서드의 이름을 분석해서 jpql 쿼리를 실행한다.

예) List<School> findByNameOrderByPlaceAsc(String name);

name인 School를 찾되 place 를 기준으로 오름차순 정렬

# 03 정렬과 페이징 처리

## 페이징 처리

페이징: 데이터베이스의 레코드를 개수로 나워서 페이지를 구분하는 것

JPA에서는 페이징 처리를 하기 위해 Page, Pageable를 사용한다.

Repository에서의 정의 예시

```
Page<School> findByName(String name, Pageable pageable);
```

# 03 정렬과 페이징 처리

## 페이징 처리

요청 예시

```
Page<School> schoolPage = shcoolRepository.findByName("동미대", PageRequest.of(0 ,2));
```

PageRequest.of() 메서드

of(int page, int size)
- int page: 페이지 번호
- int size: 페이지당 데이터 개수

of(int page, int size, Sort sort)
- Sort sort: 정렬

# 04 출처

https://adjh54.tistory.com/334
https://hanhyx.tistory.com/35
https://east-star.tistory.com/10
https://qpdh.tistory.com/211
https://velog.io/@kimdy0915/Spring-Data-JPA%EB%A5%BC-%ED%99%9C%EC%9A%A9%ED%95%9C-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EC%A0%95%EB%A0%AC-%EA%B5%AC%ED%98%84JPA-method-Pageable-Page