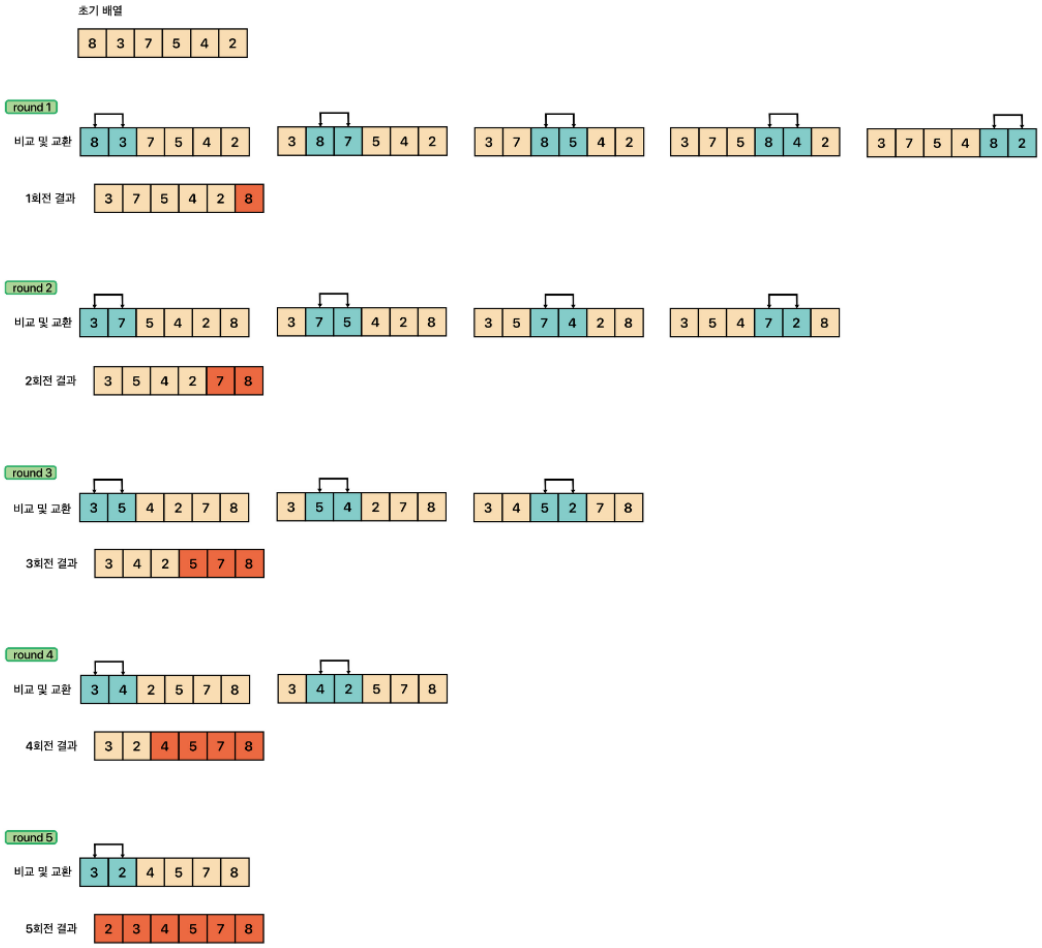
버블정렬

작성일시	@2023년 6월 15일 오전 1:15
강의 번호	Algorithm
유형	
자료	
복습	☐
Spring Framwork	

💡 인접한 두 요소의 크기를 '비교'하면서 찾는 '비교 정렬'이다.

또한 데이터 외의 추가적인 공간을 필요로 하지 않기 때문에 '제자리 정렬(in-place sort)이기도 하다.
데이터를 서로 '교환'하는 과정에서 임시 변수를 필요로 하지만 충분히 무시할 만큼 적은 양이기에 제자리 정렬로 보아도 무방하다.
선택 정렬과는 달리 거품 정렬은 앞에서부터 차례대로 비교하기 때문에 '안정 정렬'이기도 하다.

> https://st-lab.tistory.com/195



총 라운드는 배열 크기 -1번 진행된다.
각 라운드별 비교 횟수는 배열 크기 - 라운드 횟수 만큼 비교한다.

구현 코드 1번

```
static void swap(int[]a , int idx1,int idx2) {
  int tmp = a[idx1];
  a[idx1] = a[idx2];
  a[idx2] = tmp;
}

static void bubbleSort(int[] a, int n) {
  for(int i=0; i<n-1; i++) {  //배열의 첫번째부터 가장 작은 수를 집어넣는다.(n-1까지 하는 이유 : 맨 마지막은 알아서 가장 큰 숫자로 정렬)
    for(int j =n-1; j>i;j--)  //검색은 배열의 뒤부터 차례대로 교체
      if(a[j-1] > a[j]) swap(a, j-1, j);  //a[j]보다 a[j-1]이 더 작으면 스왑
  }
```

구현 코드 2번

```
static void swap(int[]a , int idx1,int idx2) {
    int tmp = a[idx1];
    a[idx1] = a[idx2];
    a[idx2] = tmp;
  }

  static void bubbleSort(int[] a, int n) {
    for(int i=0; i<n-1; i++) {  //배열의 첫번째부터 가장 작은 수를 집어넣는다.(n-1까지 하는 이유 : 맨 마지막은 알아서 가장 큰 숫자로 정렬)
      int change =0;
      for(int j =n-1; j>i;j--)  //검색은 배열의 뒤부터 차례대로 교체
        if(a[j-1] > a[j]) {
          swap(a, j-1, j);  //a[j-1]보다 a[j]가 더 작으면 스왑
          change +=1;
        }
      if(change == 0) break;
    }
```

💡 1번과 차이점 : change가 교환유무를 판단한다.
change가 0인 경우 : 더 이상 교환이 필요없음으로 판단하고 수행 종료
change가 1이상인 경우 : 교환이 필요하다 판단하여 반복문 수행

구현 코드 3번

```
static void swap(int[]a , int idx1,int idx2) {
    int tmp = a[idx1];
    a[idx1] = a[idx2];
    a[idx2] = tmp;
  }

  static void bubbleSort(int[] a, int n) {
      int k=0;        //k : 시작점
      while(k< n-1) {
        int last = n -1;    //마지막으로 교환이이루어진 위치

      for(int j =n-1; j>k ;j--) { //배열의 가장 뒤부터 검색 후  차례대로 교체
        if(a[j-1] > a[j]) {
          swap(a, j-1, j);  //a[j-1]보다 a[j]가 더 작으면 스왑
          last = j;
          }//if
        k  = last;
        }//for j
      }//while

  }//bubbleSort
```

💡 구현 코드 3번의 차이점

while문이 반복하면서 배열의 전체 크기를 교환이 필요한 부분만 돌도록 수행한다.
k는 교환이 이루어진 마지막 지점을 마지막으로 지정한다.

구현 코드 4번(가장 큰 숫자가 가장 앞에 있는 경우) - 칵테일 정렬/세이커 정렬

```
package bubble;

import java.util.Scanner;

public class Q5 {
  static void swap(int[] a, int idx1, int idx2) {
    int tmp= a[idx1];
    a[idx1] = a[idx2];
    a[idx2] = tmp;
  }

  static void bubbleSortOdd(int[] a,int lx) {
    for(int i=0; i<lx-1; i+=2) {
      int change = 0;
      for( int j= lx-1; j>i ; j--) {
        if(a[j-1]>a[j]) swap(a, j-1, j);
        change +=1;
      }
      if(change ==0) break;
    }
  }
  static void bubbleSortEven(int[] a,int lx) {
    for( int j= lx-1; j>1; j--) {
      int change = 0;
      for(int i=1; i<lx-1; i+=2) {
        if(a[i]>a[i+1]) swap(a, i, i+1);
        change +=1;
      }
      if(change ==0) break;
    }
  }


  public static void main(String[] args) {
    /*
    이 배열의 두 번째 요소부터는 정렬은 되어 있지만 버전3의 버블 정렬 알고리즘을 사용해도 빠른 시간 안에 해결 할 수 없습니다.
    왜냐하면 가장 큰수가 가장 앞에 있기 때문입니다.(1회의 패스를 거쳐도 한 칸씩만 뒤로 옮겨짐)
    그래서 홀 수 번째 패스는 가장 작은 요소를 맨 앞으로 옮기고 짝수 번째 패스는 가장 큰 요소를 맨 뒤로 옮기는 방식을 사용하면 더 적은 횟
    수로 실행할 수 있습니다.
    해당 알고리즘은 양방향 버블 정렬, 칵테일 정렬, 세이커 정렬이라는 이름으로 알려져 있다.
    양방향 버블 정렬을 수행하는 프로그램을 작성하세요*/

    //짝수일때 교환과
    //홀수일때 교환을 따로 사용한다.

    Scanner stdIn = new Scanner(System.in);

    System.out.println("단순교환정렬(버블 정렬)");
    System.out.print("요솟수 : ");
    int nx = stdIn.nextInt();
    int[] x = new int[nx];

    for (int i = 0; i < nx; i++) {
      System.out.print("x[" + i + "] : ");
      x[i] = stdIn.nextInt();
    }
    for(int i=0; i<nx; i++) {
    bubbleSortEven(x, nx);
    bubbleSortOdd(x, nx);
    }
    System.out.println("오름차순으로 정렬하였습니다.");
    for (int i = 0; i < nx; i++)
      System.out.println("x[" + i + "]=" + x[i]);
  }

  }
```