# 단순 삽입 정렬

| | |
|---|---|
| 작성일시 | @2023년 6월 15일 오전 1:38 |
| 강의 번호 | Algorithm |
| 유형 | |
| 자료 | |
| 복습 | ☐ |
| Spring Framwork | |

💡 비교하고자 하는 타겟과 그 이전의 원소들과 비교하며 자리를 ‘교환’하는 정렬
비교하면서 찾기 때문에 ‘비교 정렬’
정렬의 대상이 되는 데이터 외에 추가적인 공간을 필요로 하지 않기 때문에 ‘제자리 정렬’
선택 정렬과 달리 ‘안정 정렬’

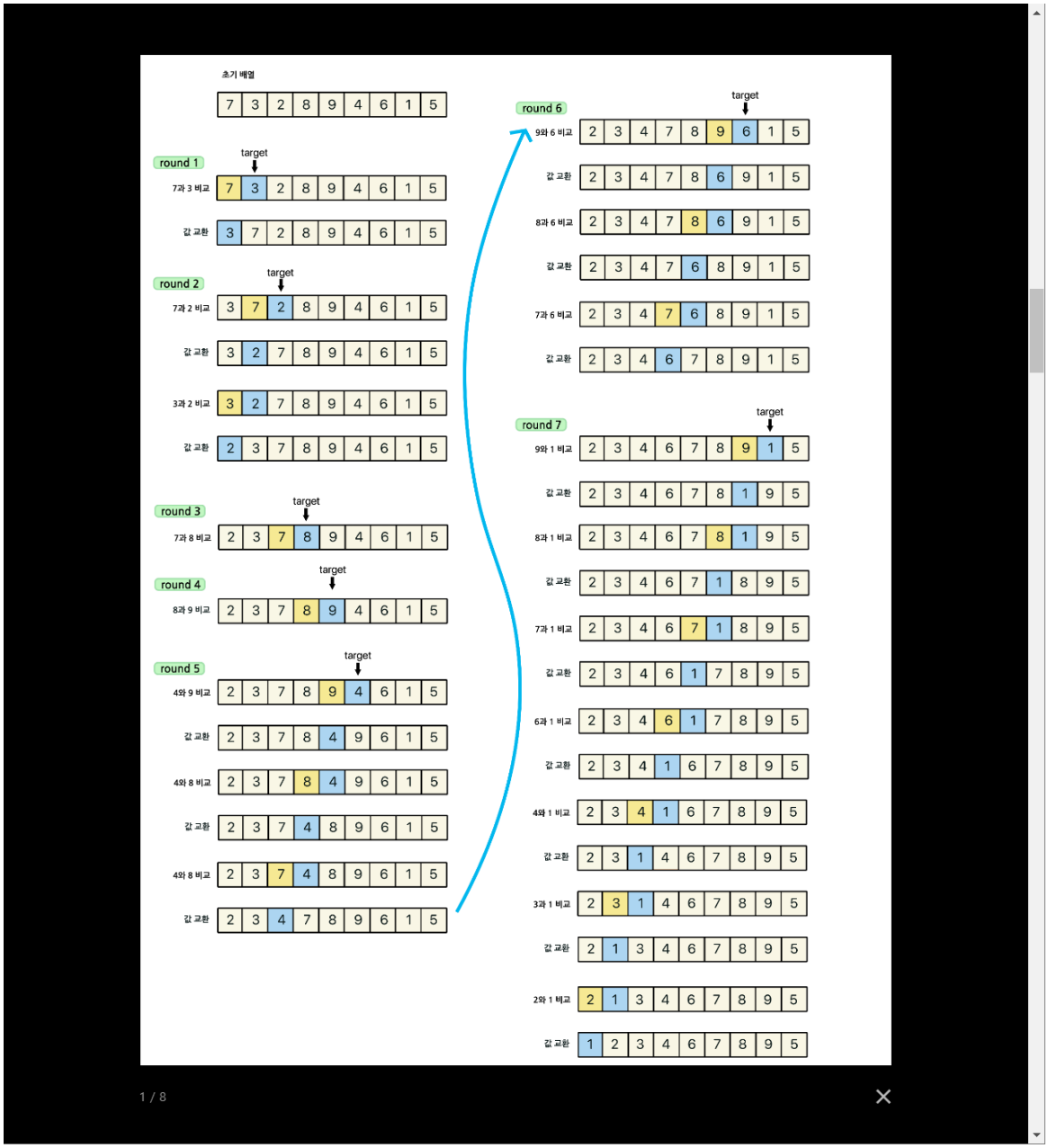
원리
해당 원소가 위치할 곳을 탐색하고 해당 위치에 삽입한다.
1. 현재 타겟이 되는 숫자와 이전 위치에 있는 원소들을 비교(첫 번째 타겟은 두 번째 원소부터 시작한다.)
2. 타겟이 되는 숫자가 이전 위치에 있던 원소보다 작다면 위치를 서로 교환한다.
3. 그 다음 타겟을 찾아 위와 같은 방법으로 반복한다.

장점
추가적인 메모리 소비가 작다.
거의 정렬된 경우 매우 효율적이다. 최선의 경우$O(N)$의 시간복잡도를 갖는다.
안장정렬이 가능하다.

단점
역순에 가까울 수록 비효율적이다. 최악의 경우$O(N^2)$의 시간 복잡도를 갖는다.
데이터의 상태에 따라 성능 편차가 매우 크다.

삽입 정렬의 경우 거의 정렬된 배열에서 좋은 성능을 보이기 때문에 실제로 병합 정렬과 삽입 정렬을 혼합한 Tim Sort(팀 정렬)이 있다. 언어에서 라이브러리로 정렬 알고리즘에 적용하고 있는 언어도 있다.

그리고 삽입 정렬을 변형하여 만들어낸 Shell Sort도 있다.


초기 배열
7 3 2 8 9 4 6 1 5
round 1
target
7과 3 비교
7 3 2 8 9 4 6 1 5
값 교환
3 7 2 8 9 4 6 1 5
round 2
target
7과 2 비교
3 7 2 8 9 4 6 1 5
값 교환
3 2 7 8 9 4 6 1 5
3과 2 비교
3 2 7 8 9 4 6 1 5
값 교환
2 3 7 8 9 4 6 1 5
round 3
target
7과 8 비교
2 3 7 8 9 4 6 1 5
round 4
target
8과 9 비교
2 3 7 8 9 4 6 1 5
round 5
target
4와 9 비교
2 3 7 8 9 4 6 1 5
값 교환
2 3 7 8 4 9 6 1 5
4와 8 비교
2 3 7 8 4 9 6 1 5
값 교환
2 3 7 4 8 9 6 1 5
4와 8 비교
2 3 7 4 8 9 6 1 5
값 교환
2 3 4 7 8 9 6 1 5
round 6
target
9와 6 비교
2 3 4 7 8 9 6 1 5
값 교환
2 3 4 7 8 6 9 1 5
8과 6 비교
2 3 4 7 8 6 9 1 5
값 교환
2 3 4 7 6 8 9 1 5
7과 6 비교
2 3 4 7 6 8 9 1 5
값 교환
2 3 4 6 7 8 9 1 5
round 7
target
9와 1 비교
2 3 4 6 7 8 9 1 5
값 교환
2 3 4 6 7 8 1 9 5
8과 1 비교
2 3 4 6 7 8 1 9 5
값 교환
2 3 4 6 7 1 8 9 5
7과 1 비교
2 3 4 6 7 1 8 9 5
값 교환
2 3 4 6 1 7 8 9 5
6과 1 비교
2 3 4 6 1 7 8 9 5
값 교환
2 3 4 1 6 7 8 9 5
4와 1 비교
2 3 4 1 6 7 8 9 5
값 교환
2 3 1 4 6 7 8 9 5
3과 1 비교
2 3 1 4 6 7 8 9 5
값 교환
2 1 3 4 6 7 8 9 5
2와 1 비교
2 1 3 4 6 7 8 9 5
값 교환
1 2 3 4 6 7 8 9 5
1 / 8


https://st-lab.tistory.com/179

## 코드 구현 1

```
static void insertionSort(int[] a, int n) {
  for(int i=1; i<n;i++) {           // 0  1  2 3   4  5  6
    int tmp=a[i];             //{22,5,11,32,120,68,70};
    int j;
    for(j=i; j>0 && a[j-1] > tmp; j--){
      a[j] = a[j-1];
    }
    a[j] = tmp;
  }//for

  }//insertionSort
```

# Shell Sort(셸 정렬) 코드 구현

```
버전 1

static void shellSort(int[] a, int n) {
    for(int h= n/2; h>0; h/=2)        //h=3  배열을 반씩 나눈다.
//      ================================================
      for(int i=h ; i <n; i++) {      //i=3,4,5,6,7   나눈배열에서 1씩 증가하고 배열의 길이까지
        int j;
        int tmp= a[i];            //tmp = a[3...7]  임시변수에는 배열의 시작값을 담는다.
//        ================================================================
        for(j=i-h; j>=0 && a[j] > tmp; j-=h) //j=0...4
          a[j+h] = a[j];          //a[3...7] = a[0...4]
//        ================================================================
        a[j+h]  = tmp;            //a[3...7]  =tmp;
      }
//    ============================================================
  }
```

# 코드구현2 (더 정확함)

```
static void shellSort(int[] a, int n) {
  int h;
      for(h=1;h<n/9;h=h*3+1)                //h는 3n+1씩 증가한다.
    {;}

  for(; h>0; h/=3) {                          // h를 3으로 계속 나눔
    for(int i=h; i<n ; i++) {                 // 삽입 정렬 수행
      int j;
      int tmp= a[i];
      for (j=i-h; j>=0 && a[j] > tmp; j-=h)   // h만큼의 거리에 있는 값과 함께 정렬
        { a[j+h] = a[j];}                     //4,0 5,1 6,2 7,3 8,4...
    a[j+h]  = tmp;
    }


  }
}
```