선택정렬 (Selection Sort)

진행 순서

1. 배열 전체 중 최대 or 최소값을 찾는다.
2. 찾은 값을 배열의 맨 처음 위치의 값과 SWAP한다.
3. SWAP했던 위치를 제외한 이후 위치부터 위 과정을 반복한다.

코드

```
void selectionSort() {
    int arr[5] = {5, 4, 3, 2, 1};

    for(int i=0; i<4; ++i){ // 정렬할 원소를 순서대로 배치할 idx po
        int minIdx = i;
        for(int j=i+1; j<5; ++j){
            if(arr[j] < arr[minIdx]){
                minIdx = j;
            }
        }
        int tmp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = tmp;
    }
}
```

시간 복잡도

- 1회전 : 1 ~ (n-1) ⇒ n-1
- 2회전 : 2 ~ (n-1) ⇒ n-2
- ...
- n-1회전 : (n-1) ~ (n-1) ⇒ 1

(n-1) + (n-2) + ... + 2 + 1 ⇒ n(n-1)/2 ⇒ O(N^2)

최선	평균	최악
$O(N^2)$	$O(N^2)$	$O(N^2)$

공간 복잡도

> 한개의 배열 안에서 수행

$$O(N)$$

장점

- 구현 단순
- 정렬을 위해 비교 횟수는 많지만,
  버블 정렬에 비해 실제 교환 횟수는 적기 때문에,
  "교환이 많이 일어나야"하는 자료 상태에서 비교적 효율적
- 정렬하고자 하는 배열안에서 정렬 교환이 발생하므로,
  다른 메모리 공간을 필요로 하지 않는다.

단점

- 최선, 평균, 최악이 O(N^2)으로 비효율
- 정렬되어 있지 않은 원소가 많으면 SWAP이 많이 발생
- unstable sort
  - 동일한 정렬 기준(같은 크기의 원소)을 가진 것은 정렬 후 위치가 달라 질 수 있다.