**정렬**

* 선택 정렬(selection sort) 알고리즘을 이해한다.
* 선택 정렬(selection sort) 알고리즘을 c언어로 구현한다.
* 선택 정렬(selection sort) 알고리즘의 특징
* 선택 정렬(selection sort) 알고리즘의 시간복잡도를 이해한다.

들어가기 전

* 오름차순을 기준으로 정렬한다.

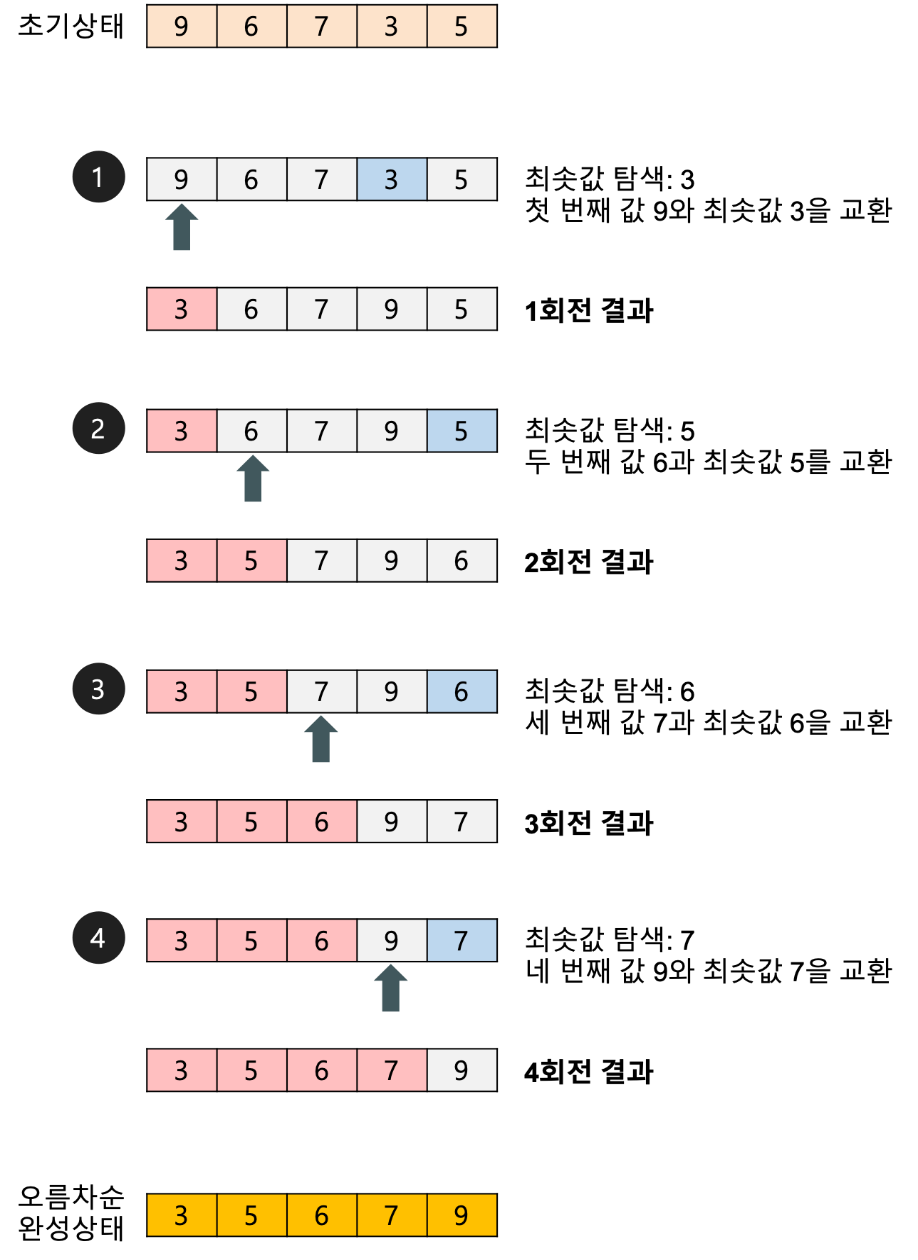
**선택 정렬(selection sort) 알고리즘 개념 요약**

* 제자리 정렬(in-place sorting) 알고리즘의 하나
  + 입력 배열(정렬되지 않은 값들) 이외에 다른 추가 메모리를 요구하지 않는 정렬 방법
* 해당 순서에 원소를 넣을 위치는 이미 정해져 있고, 어떤 원소를 넣을지 선택하는 알고리즘
  + 첫 번째 순서에는 첫 번째 위치에 가장 최솟값을 넣는다.
  + 두 번째 순서에는 두 번째 위치에 남은 값 중에서의 최솟값을 넣는다.
  + …
* 과정 설명
  + 주어진 배열 중에서 최솟값을 찾는다.
  + 그 값을 맨 앞에 위치한 값과 교체한다(패스(pass)).
  + 맨 처음 위치를 뺀 나머지 리스트를 같은 방법으로 교체한다.
  + 하나의 원소만 남을 때까지 위의 1~3 과정을 반복한다.

**선택 정렬(selection sort) 알고리즘의 구체적인 개념**

* 선택 정렬은 첫 번째 자료를 두 번째 자료부터 마지막 자료까지 차례대로 비교하여 가장 작은 값을 찾아 첫 번째에 놓고, 두 번째 자료를 세 번째 자료부터 마지막 자료까지와 차례대로 비교하여 그 중 가장 작은 값을 찾아 두 번째 위치에 놓는 과정을 반복하며 정렬을 수행한다.
* **1회전을 수행하고 나면 가장 작은 값의 자료가 맨 앞에 오게 되므로** 그 다음 회전에서는 두 번째 자료를 가지고 비교한다. 마찬가지로 3회전에서는 세 번째 자료를 정렬한다.

**선택 정렬(selection sort) 알고리즘의 예제**

* 배열에 9, 6, 7, 3, 5가 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.
* 
* 1회전:
  + 첫 번째 자료 9를 두 번째 자료부터 마지막 자료까지와 비교하여 가장 작은 값을 첫 번째 위치에 옮겨 놓는다. 이 과정에서 자료를 4번 비교한다.
* 2회전:
  + 두 번째 자료 6을 세 번째 자료부터 마지막 자료까지와 비교하여 가장 작은 값을 두 번째 위치에 옮겨 놓는다. 이 과정에서 자료를 3번 비교한다.
* 3회전:
  + 세 번째 자료 7을 네 번째 자료부터 마지막 자료까지와 비교하여 가장 작은 값을 세 번째 위치에 옮겨 놓는다. 이 과정에서 자료를 2번 비교한다.
* 4회전:
  + 네 번째 자료 9와 마지막에 있는 7을 비교하여 서로 교환한다.

**선택 정렬(selection sort) c언어 코드**

# include <stdio.h>

# define SWAP(x, y, temp) ( (temp)=(x), (x)=(y), (y)=(temp) )

# define MAX\_SIZE 5

// 선택 정렬

void selection\_sort(int list[], int n){

int i, j, least, temp;

// 마지막 숫자는 자동으로 정렬되기 때문에 (숫자 개수-1) 만큼 반복한다.

for(i=0; i<n-1; i++){

least = i;

// 최솟값을 탐색한다.

for(j=i+1; j<n; j++){

if(list[j]<list[least])

least = j;

}

// 최솟값이 자기 자신이면 자료 이동을 하지 않는다.

if(i != least){

SWAP(list[i], list[least], temp);

}

}

}

void main(){

int i;

int n = MAX\_SIZE;

int list[n] = {9, 6, 7, 3, 5};

// 선택 정렬 수행

selection\_sort(list, n);

// 정렬 결과 출력

for(i=0; i<n; i++){

printf("%d\n", list[i]);

}

}

**선택 정렬(selection sort) 알고리즘의 특징**

* 장점
  + 자료 이동 횟수가 미리 결정된다.
* 단점
  + 안정성을 만족하지 않는다.
  + 즉, 값이 같은 레코드가 있는 경우에 상대적인 위치가 변경될 수 있다.

**선택 정렬(selection sort)의 시간복잡도**

시간복잡도를 계산한다면

* 비교 횟수
  + 두 개의 for 루프의 실행 횟수
  + 외부 루프: (n-1)번
  + 내부 루프(최솟값 찾기): n-1, n-2, … , 2, 1 번
* 교환 횟수
  + 외부 루프의 실행 횟수와 동일. 즉, 상수 시간 작업
  + 한 번 교환하기 위하여 3번의 이동(SWAP 함수의 작업)이 필요하므로 3(n-1)번
* T(n) = (n-1) + (n-2) + … + 2 + 1 = n(n-1)/2 = **O(n^2)**

**정렬 알고리즘 시간복잡도 비교**



* 단순(구현 간단)하지만 비효율적인 방법
  + 삽입 정렬, **선택 정렬**, 버블 정렬
* 복잡하지만 효율적인 방법
  + 퀵 정렬, 힙 정렬, 합병 정렬, 기수 정렬