algorithm 삽입정렬 sort

[알고리즘] 삽입 정렬(insertion sort)이란

heejeong Kwon 06 May 2018

손안의 카드를 정렬하는 방법과 유사한 알고리즘

Goal

> - 삽입 정렬(insertion sort) 알고리즘을 이해한다.
> - 삽입 정렬(insertion sort) 알고리즘을 c언어로 구현한다.
> - 삽입 정렬(insertion sort) 알고리즘의 특징
> - 삽입 정렬(insertion sort) 알고리즘의 시간복잡도를 이해한다.

들어가기 전

- 오름차순을 기준으로 정렬한다.

삽입 정렬(insertion sort) 알고리즘 개념 요약

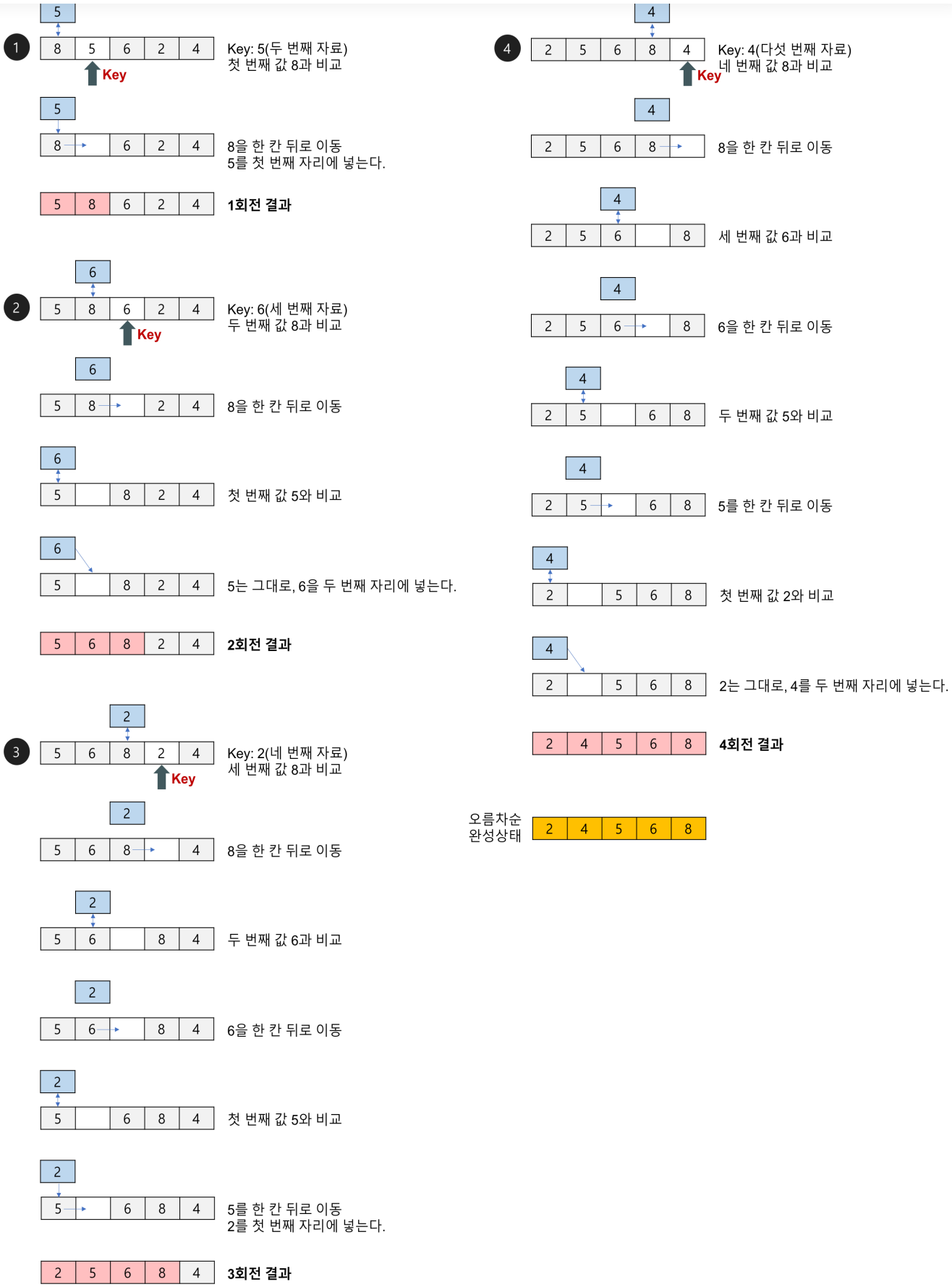
- 손안의 카드를 정렬하는 방법과 유사하다.
  - 새로운 카드를 기존의 정렬된 카드 사이의 올바른 자리를 찾아 삽입한다.
  - 새로 삽입될 카드의 수만큼 반복하게 되면 전체 카드가 정렬된다.
- 자료 배열의 모든 요소를 **앞에서부터 차례대로 이미 정렬된 배열 부분과 비교** 하여, 자신의 위치를 찾아 삽입함으로써 정렬을 완성하는 알고리즘
- 매 순서마다 해당 원소를 삽입할 수 있는 위치를 찾아 해당 위치에 넣는다.

삽입 정렬(insertion sort) 알고리즘의 구체적인 개념

- 즉, 두 번째 자료는 첫 번째 자료, 세 번째 자료는 두 번째와 첫 번째 자료, 네 번째 자료는 세 번째, 두 번째, 첫 번째 자료와 비교한 후 자료가 삽입될 위치를 찾는다. 자료가 삽입될 위치를 찾았다면 그 위치에 자료를 삽입하기 위해 자료를 한 칸씩 뒤로 이동시킨다.
- **처음 Key 값은 두 번째 자료부터 시작한다**.

삽입 정렬(insertion sort) 알고리즘의 예제

- 배열에 8, 5, 6, 2, 4가 저장되어 있다고 가정하고 자료를 오름차순으로 정렬해 보자.
-



- 1회전: 두 번째 자료인 5를 Key로 해서 그 이전의 자료들과 비교한다.
  - Key 값 5와 첫 번째 자료인 8을 비교한다. 8이 5보다 크므로 8을 5자리에 넣고 Key 값 5를 8의 자리인 첫 번째에 기억시킨다.

에 기억시킨다.

- Key 값 6과 첫 번째 자료인 5를 비교한다. 5가 Key 값보다 작으므로 Key 값 6을 두 번째 자리에 기억시킨다.

- 3회전: 네 번째 자료인 2를 Key 값으로 해서 그 이전의 자료들과 비교한다.
  - Key 값 2와 세 번째 자료인 8을 비교한다. 8이 Key 값보다 크므로 8을 2가 있던 네 번째 자리에 기억시킨다.
  - Key 값 2와 두 번째 자료인 6을 비교한다. 6이 Key 값보다 크므로 6을 세 번째 자리에 기억시킨다.
  - Key 값 2와 첫 번째 자료인 5를 비교한다. 5가 Key 값보다 크므로 5를 두 번째 자리에 넣고 그 자리에 Key 값 2를 기억시킨다.
- 4회전: 다섯 번째 자료인 4를 Key 값으로 해서 그 이전의 자료들과 비교한다.
  - Key 값 4와 네 번째 자료인 8을 비교한다. 8이 Key 값보다 크므로 8을 다섯 번째 자리에 기억시킨다.
  - Key 값 4와 세 번째 자료인 6을 비교한다. 6이 Key 값보다 크므로 6을 네 번째 자리에 기억시킨다.
  - Key 값 4와 두 번째 자료인 5를 비교한다. 5가 Key 값보다 크므로 5를 세 번째 자리에 기억시킨다.
  - Key 값 4와 첫 번째 자료인 2를 비교한다. 2가 Key 값보다 작으므로 4를 두 번째 자리에 기억시킨다.

삽입 정렬(insertion sort) c언어 코드

```c
# include <stdio.h>
# define MAX_SIZE 5

// 삽입 정렬
void insertion_sort(int list[], int n){
  int i, j, key;

  // 인덱스 0은 이미 정렬된 것으로 볼 수 있다.
  for(i=1; i<n; i++){
    key = list[i]; // 현재 삽입될 숫자인 i번째 정수를 key 변수로 복사

    // 현재 정렬된 배열은 i-1까지이므로 i-1번째부터 역순으로 조사한다.
    // j 값은 음수가 아니어야 되고
    // key 값보다 정렬된 배열에 있는 값이 크면 j번째를 j+1번째로 이동
    for(j=i-1; j>=0 && list[j]>key; j--){
      list[j+1] = list[j]; // 레코드의 오른쪽으로 이동
    }
```

```
  }
}

void main(){
  int i;
  int n = MAX_SIZE;
  int list[n] = {8, 5, 6, 2, 4};

  // 삽입 정렬 수행
  insertion_sort(list, n);

  // 정렬 결과 출력
  for(i=0; i<n; i++){
    printf("%d\n", list[i]);
  }
}
```

삽입 정렬(insertion sort) 알고리즘의 특징

- 장점
  - 안정한 정렬 방법
  - 레코드의 수가 적을 경우 알고리즘 자체가 매우 간단하므로 다른 복잡한 정렬 방법보다 유리할 수 있다.
  - 대부분위 레코드가 이미 정렬되어 있는 경우에 매우 효율적일 수 있다.
- 단점
  - 비교적 많은 레코드들의 이동을 포함한다.
  - 레코드 수가 많고 레코드 크기가 클 경우에 적합하지 않다.

삽입 정렬(insertion sort)의 시간복잡도

시간복잡도를 계산한다면

- 최선의 경우
  - 비교 횟수
    - 이동 없이 1번의 비교만 이루어진다.
    - 외부 루프: (n-1)번
  - Best T(n) = O(n)

- 외부 루프 안의 각 단계마다 1번의 비교 수행
- 외부 루프: (n-1) + (n-2) + ... + 2 + 1 = n(n-1)/2 = O(n^2)
- 교환 횟수
  - 외부 루프의 각 단계마다 (i+2)번의 이동 발생
  - n(n-1)/2 + 2(n-1) = (n^2+3n-4)/2 = **O(n^2)**
- Worst T(n) = **O(n^2)**

정렬 알고리즘 시간복잡도 비교

Name	Best	Avg	Worst	Run-time(정수 60,000개) 단위: sec
삽입정렬	n	n^2	n^2	7.438
선택정렬	n^2	n^2	n^2	10.842
버블정렬	n^2	n^2	n^2	22.894
셸 정렬	n	$n^{1.5}$	n^2	0.056
퀵 정렬	$n\log_2 n$	$n\log_2 n$	n^2	0.014
힙 정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.034
병합정렬	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	0.026

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

관련된 Post

- 선택 정렬(selection sort): 선택 정렬(selection sort) 을 참고하시기 바랍니다.
- 버블 정렬(bubble sort): 버블 정렬(bubble sort) 을 참고하시기 바랍니다.
- 셸 정렬(shell sort): 셸 정렬(shell sort) 을 참고하시기 바랍니다.
- 합병 정렬(merge sort): 합병 정렬(merge sort) 을 참고하시기 바랍니다.
- 퀵 정렬(quick sort): 퀵 정렬(quick sort) 을 참고하시기 바랍니다.
- 힙 정렬(heap sort): 힙 정렬(heap sort) 을 참고하시기 바랍니다.

- https://jongmin92.github.io/2017/11/06/Algorithm/Concept/basic-sort/
- 삽입 정렬 - 위키백과
- C언어로 쉽게 풀어 쓴 자료구조

HeeJeong Kwon
computer programmer

[알고리즘] 선택 정렬 (selection sort)이란
Goal 선택 정렬(selection sort) 알고리즘을 이해한다. 선택 정렬(sele...

[알고리즘] 버블 정렬 (bubble sort)이란
Goal 버블 정렬(bubble sort) 알고리즘을 이해한다. 버블 정렬(bubble ...