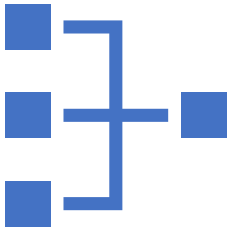
자료구조 알고리즘
: Bubble sort, insertion sort, quick sort

이지훈

목차



What is sort?

Sort

Bubble sort

Insertion sort

Quick sort

Sort.c

Bubble sort

Insertion sort

Quick sort

Sort algorithm

N개의 숫자가 입력으로 주어졌을 때, 지정한 기준에 맞게
정렬(sort)하여 출력하는 알고리즘.

선택 정렬, 삽입 정렬, 퀵 정렬 등 여러가지의 알고리즘이 있으며,
각각의 알고리즘들의 시간 복잡도와 공간 복잡도가 다르다.

Sort algorithm

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

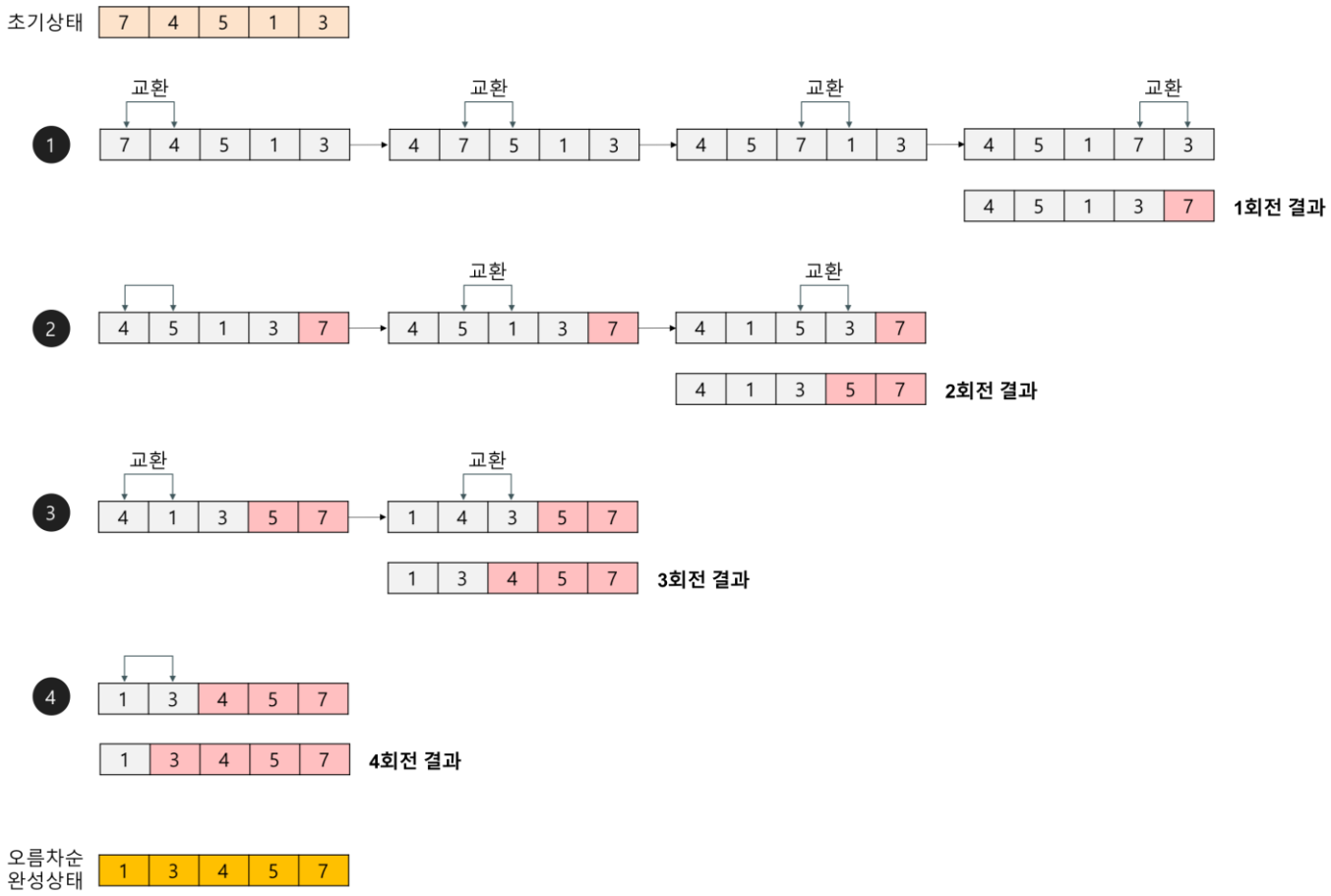
Bubble sort

버블 정렬 : 자료 배열에서 서로 인접한 두 요소를 비교하여 정렬하는 알고리즘
크기가 순서대로 되어있지 않으면(오름차순, 내림차순) 서로 교환한다.

장점 : 알고리즘 구현이 매우 간단하다.

단점 : 가장 왼쪽에서 가장 오른쪽으로 움직이기 위해선 모든 다른 요소들과 교환되어야 한다.
이미 최종 정렬 위치에 있더라도 교환되는 일이 일어난다.

Bubble sort


초기상태
7 4 5 1 3
교환
1
4 5 1 3 7
1회전 결과
2
2회전 결과
3
3회전 결과
4
4회전 결과
오름차순
완성상태
1 3 4 5 7

Bubble sort.c

```c
void bubble_sort(int arr[], int max) {
    int i, j, tmp;
    for (i = 0; i < max - 1; i++) {

        for (j = 0; j < max - 1; j++) {

            if (arr[j] < arr[j + 1]) {
                tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
        }
    }
}
```

Insertion sort

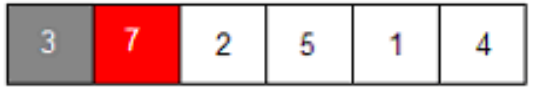
삽입 정렬 : 자료 배열의 모든 요소를 앞에서부터 차례대로 비교하며 자신의 위치를 찾고 그 위치에 삽입하는 정렬 알고리즘.

장점 : 알고리즘 구현이 간단하다.
어느정도 정렬된 배열에 효율적일 수 있다.

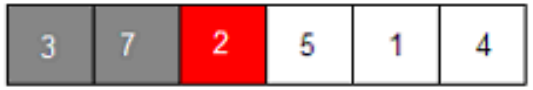
단점 : 배열의 이동이 비교적 많은 편이다.
배열의 크기가 클 경우 시간이 오래 걸린다.

Insertion sort

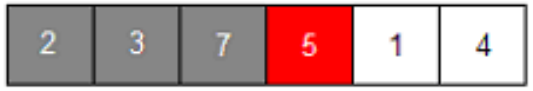
(a)



(b)



(c)



(d)

(e)

(f)

Insertion sort.c

```c
#include<stdio.h>
#include<stdlib.h>

#define MAX 10

int main(void) {
    int arr[MAX] = { 10,8,6,19,80,54,5,7,99,15 };
    int tmp, t;

    printf("정렬전 : ");
    for (int i = 0; i < MAX; i++)
        printf("%d ", *(arr+i));
    printf("₩n");

    for (int i = 1; i < MAX; i++) {
        tmp = arr[i];
        for(int j=i-1;j>=0;j--){
            if (arr[j] > tmp) {
                t = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = t;
            }
            else
                break;
        }
    }

    printf("정렬후 : ");
    for (int i = 0; i < MAX; i++)
        printf("%d ", *(arr + i));
    printf("₩n");

    return 0;
}
```

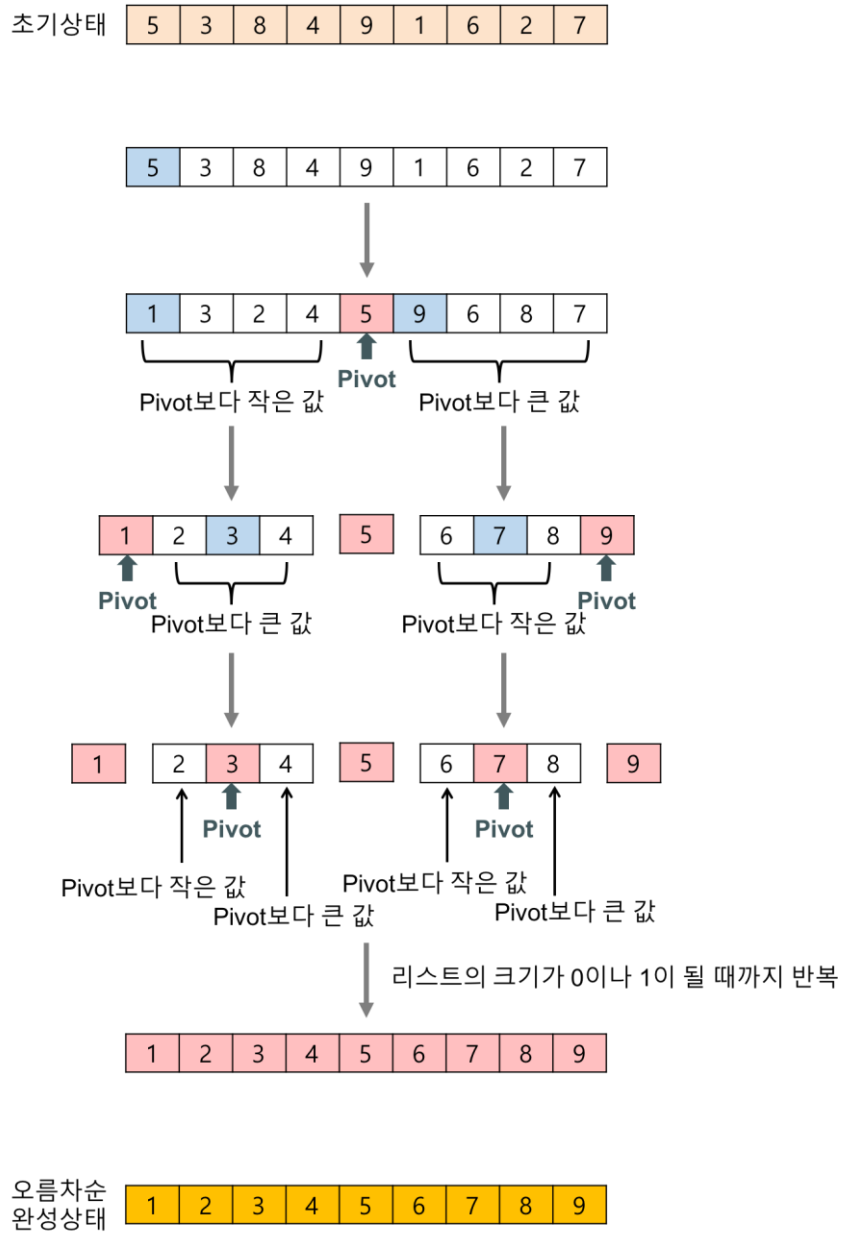
Quick sort

퀵 정렬 : 분할 정복 방법을 사용 - 문제를 작은 2개의 문제로 분리하고 각각을 해결한 다음 결과를 모아 원래의 문제를 해결하는 방식.

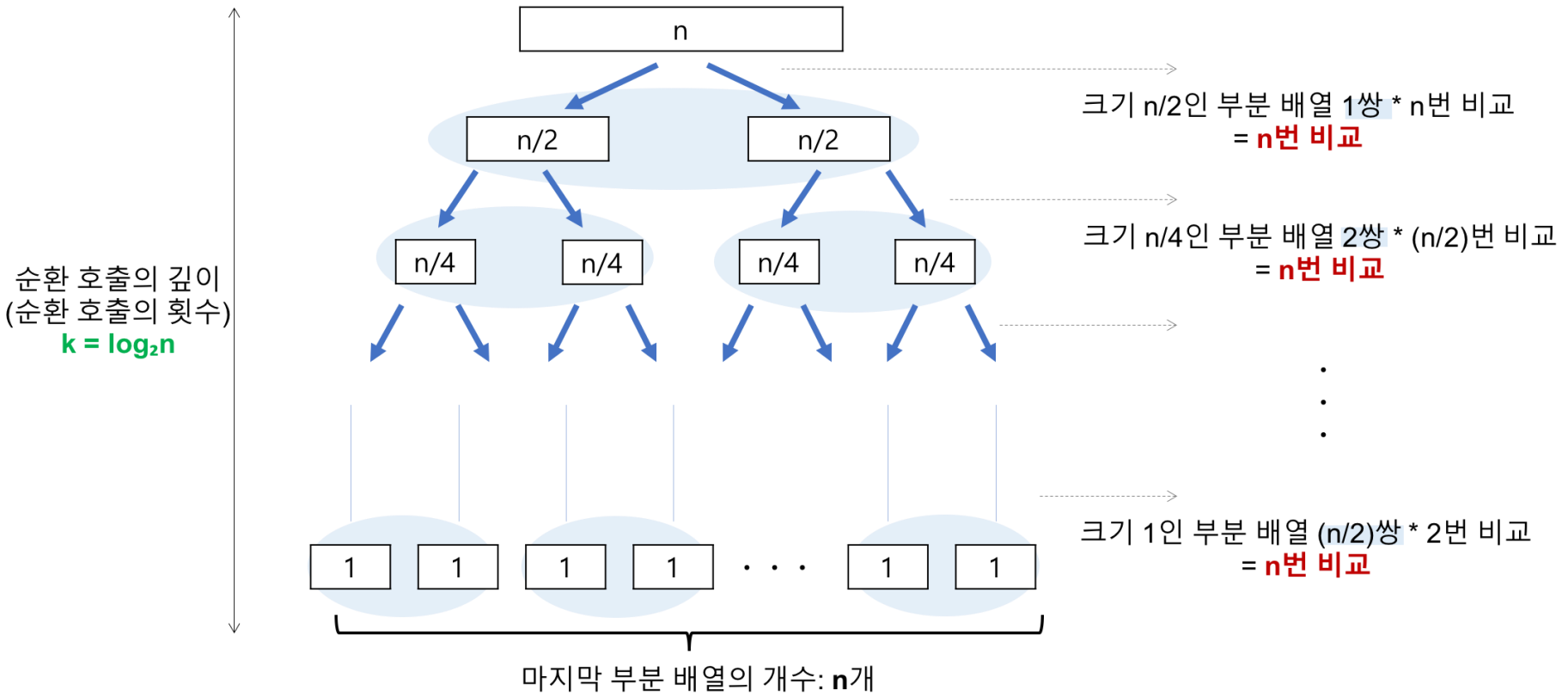
장점 : 속도가 빠르다.
추가 메모리 공간을 필요로 하지 않는다.

단점 : 정렬된 리스트에 대해서는 시간이 오히려 더 오래 걸린다.
구현하기가 다른 정렬에 비해 다소 어렵다.

Quick sort


초기상태
5 3 8 4 9 1 6 2 7
5 3 8 4 9 1 6 2 7
1 3 2 4 5 9 6 8 7
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
1 2 3 4
5
6 7 8 9
Pivot
Pivot보다 큰 값
Pivot보다 작은 값
Pivot
1
2 3 4
5
6 7 8
9
Pivot
Pivot
Pivot보다 작은 값
Pivot보다 큰 값
Pivot보다 작은 값
Pivot보다 큰 값
리스트의 크기가 0이나 1이 될 때까지 반복
1 2 3 4 5 6 7 8 9
오름차순
완성상태
1 2 3 4 5 6 7 8 9

Quick sort


n
n/2
n/2
n/4
n/4
n/4
n/4
1
1
1
1
1
1
순환 호출의 깊이
(순환 호출의 횟수)
k = log₂n
크기 n/2인 부분 배열 1쌍 * n번 비교
= n번 비교
크기 n/4인 부분 배열 2쌍 * (n/2)번 비교
= n번 비교
크기 1인 부분 배열 (n/2)쌍 * 2번 비교
= n번 비교
마지막 부분 배열의 개수: n개

Quick sort.c

```
quick(arr, 0, MAX - 1);
```

```c
void quick(int* dat, int left, int right) {
    if (left >= right)return; //데이터가 1개이하 이므로 더이상 수행할 수 없음.

    swap(&dat[left], &dat[(left + right) / 2]); //기준점을 중앙으로 옮기기 위함
    int last = left;
    for (int i = left + 1; i <= right; i++) {
        if (dat[left] > dat[i]) {
            last++; // 기준점 변경
            swap(&dat[last], (dat + i));
        }
    }

    swap((dat + left), (dat + last));

    quick(dat, left, last - 1); // 좌우로 갈라서 진행
    quick(dat, last + 1, right);
}

void swap(int* a, int* b) {
    int t;
    t = *a;
    *a = *b;
    *b = t;
    return;
}
```

Q & A

사진 출처 : https://gmlwjd9405.github.io/ 및 위키피디아