Algorithmic analysis

Overview

- Goal : insertion sort / merge sort
- pseudo code
- running time analysis 분석
- divide and conquer 학습 in merge sort

Insertion sort

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

- the sequence → array 형태로 지정
- Keys = N (갯수)
- 각각의 키가 - satellite data

특징

- 작은 수의 배열 요소를 소팅하기 좋음
- -비유
    1. 왼쪽에 빈 손 / 테이블에 카드들이 놓여있음
    2. 테이블에서 하나를 왼손에다 놓음
    3. 적절한 위치를 찾기 위해 각 카드와 이미 있는 손에 있는 카드와 비교함
    4. 왼손에 있는 카드가 정렬되고 계속 정렬됨

    즉, 손에 있는 카드를 한번에 한장씩 확인하며 정렬 되어 있는 카드 사이 적절한 위치에 삽입할때 사용하는 방법이다

삽입정렬은 요소를 삽입하기 전 대상 요소보다 큰 요소의 자리를 오른쪽으로 밀어서 빈 공간을 만들고, 만들어진 공간에 해당 요소를 삽입한다.

INSERTION-SORT(A, n)	*cost*	*times*
for $j = 2$ **to** n	c_1	n
$\quad$ $key = A[j]$	c_2	$n-1$
$\quad$ // Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
$\quad$ $i = j - 1$	c_4	$n-1$
$\quad$ **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$\quad\quad$ $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$\quad\quad$ $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$\quad$ $A[i+1] = key$	c_8	$n-1$

[Leave this on the board, but show only the pseudocode for now. We'll put in the "cost" and "times" columns later.]

1. j = key 2부터 n 만큼 정렬 탐색
2. key 부여
3. 만약 정렬해야될 애가 .. 있는 것 i=j-1

A[i]자리에 삽입해야하면 A[i+1]=A[i]로 변경 / A[i+1]=key 두개 값을 swap 해주는 부분

i=i-1 빈공간 삭제 ?

Example

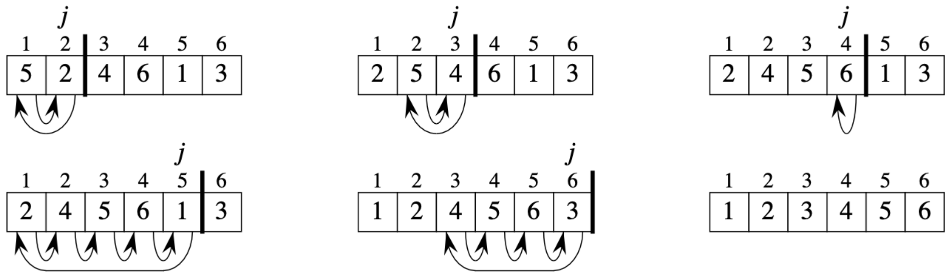

사진 기준

1. j - left hand 역할
2. 만약 j-1보다 키보다 크다면 (j보다 크다면) 옮겨준다 (left A[j] greater than A[j] i = j-1)
3. 그 다음 key 값을 증가 (j를 증가시킴)

heavy vertical line 기준으로 Iteration 영향 O / 뒤쪽은 iteration 영향을 받지 않음

key 값 뒤로는 영향을 받지 않음

correctness

- loop invariant 를 통해 적절한지를 알 수 ㅣㅇㅆ음
- A[1…j-1]까지의 루프가 필요함 → 순서대로

Correctness

We often use a ***loop invariant*** to help us understand why an algorithm gives the correct answer. Here's the loop invariant for INSERTION-SORT:

> **Loop invariant:** At the start of each iteration of the "outer" **for** loop—the loop indexed by j—the subarray $A[1 \mathinner{..} j-1]$ consists of the elements originally in $A[1 \mathinner{..} j-1]$ but in sorted order.

To use a loop invariant to prove correctness, we must show three things about it:

Initialization: It is true prior to the first iteration of the loop.

Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant—usually along with the reason that the loop terminated—gives us a useful property that helps show that the algorithm is correct.

Best case

Best case

The array is already sorted.

- Always find that $A[i] \le key$ upon the first time the **while** loop test is run (when $i = j - 1$).
- All t_j are 1.
- Running time is

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

- Can express $T(n)$ as $an + b$ for constants a and b (that depend on the statement costs c_i) $\Rightarrow T(n)$ is a *linear function* of n.

Worse case

> 최선의 조건에서는 $N-1$ 번의 비교와 0 번의 교환을 수행한다. $(n-1)+(n-2)+(n-3)+\cdots+2+1=n(n-1)/2=O(N2)$ 최악 $O(N2)$ 평균 $O(N2)$ 최선 $O(N)$

> 삽입정렬의 공간복잡도는 주어진 배열안에서 교환을 수행하는 제자리 정렬 이기 때문에 $O(N)$ 이다.

Analyzing algorithms

- Random Access machine model - RAM model
    - 동시에 수행되지 않고 하나씩 처리됨
    - in real computers:
        1. Arithmetic: add, subtract, multiply, divide, remainder, floor, ceiling). Also,
           shift left/shift right (good for multiplying/dividing by 2^k).
        2. • Data movement: load, store, copy.
        3. • Control: conditional/unconditional branch, subroutine call and return.
    - RAM model : uses integer and floating-point types.

→ the time : input 수에 비례함