Insertion Sort

Problem

- Problem
  - input: sequence of n numbers $<a_1, a_2, \ldots , a_n>$
  - output: $<a_1', a_2',\ldots, an'>$ such that $a_1' \leq a_2' \leq \ldots \leq a_n'$

- Idea
  - 왼쪽부터 차례로 정렬
  - 새로운 카드를 정렬 위치에 삽입

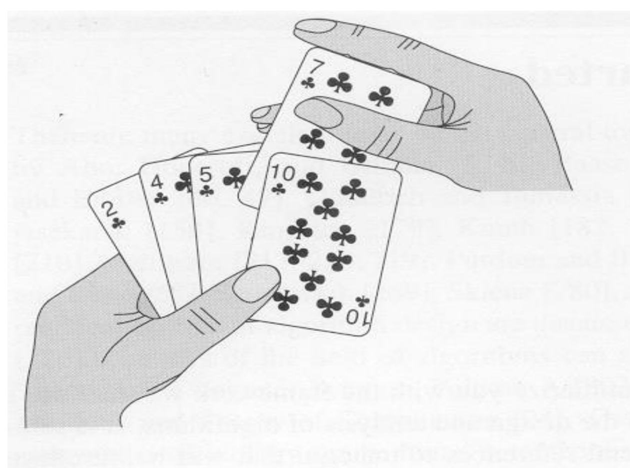


Figure 2.1 Sorting a hand of cards using insertion sort.

Algorithm

- Method: 새로운 카드를 삽입하는 방법

  (1) Selection Sort

  오른쪽 카드의 값 중 최소값을 찾아 자리 교환

  (ex) <5, 2, 4, 6, 1, 3>

  탐색 수: 20

5	**2**	**4**	**6**	**1**	**3**	(6)
1	**2**	**4**	**6**	**5**	**3**	(5)
1	**2**	**4**	**6**	**5**	**3**	(4)
1	2	**3**	**6**	**5**	**4**	(3)
1	2	3	**4**	**5**	**6**	(2)
1	2	3	4	**5**	**6**	

  (2) Insertion Sort

  왼쪽 카드의 값이 내 카드의 값보다 크면 자리 교환

  탐색 수: 11

5	2	4	6	1	3	(1)
2	**5**	4	6	1	3	(2)
2	4	**5**	6	1	3	(1)
2	**4**	**5**	**6**	1	3	(4)
1	2	**4**	**5**	**6**	3	(3)
1	2	3	4	5	6	

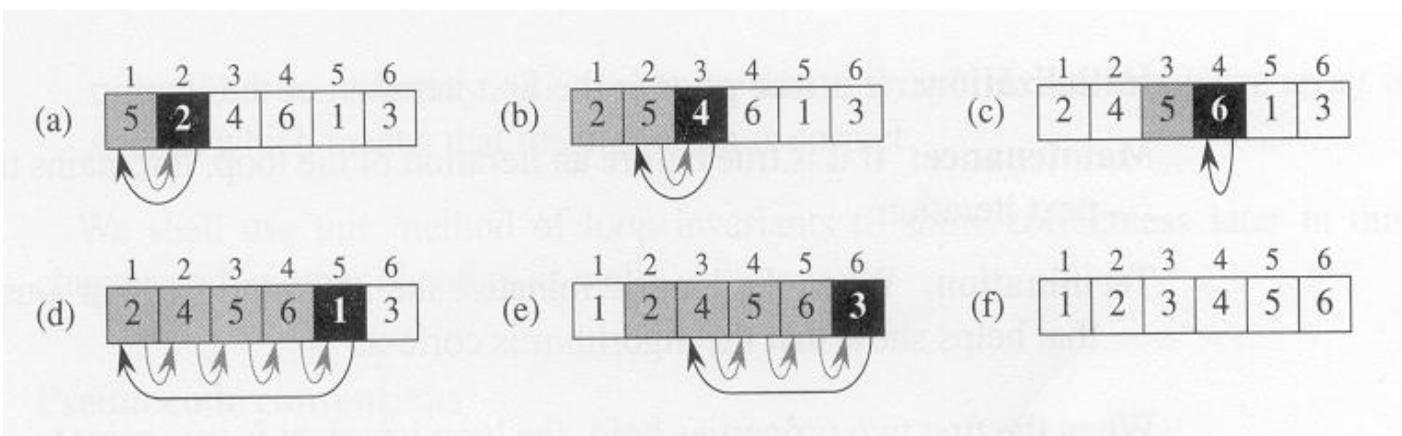
Incremental Approach

Algorithm

- Operation
  - Key 선택: 왼쪽에서 오른쪽으로 하나 씩 증가

    ⇒ Loop 1
  - Key 삽입: 선택 Key를 왼쪽의 정렬된 배열의 적당 위치에 삽입

    ⇒ Loop 2



Algorithm

- Algorithm

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

❖ Psedocode

1) Block { }: indentation
2) Loop: *for, while, repeat*, Condition: *if, then, else*
3) Comment: ▷
4) Assignment: , $i \leftarrow j$, $i \leftarrow j \leftarrow e$
5) Variables*: i, j , keys*
6) Array: *A[i]*=i-the element of array *A*,
   A[1..j]=subarray of *A*(*A[1],A[2],…,A[j]*)

Algorithm

(Q) A = < 5, 2, 4, 6, 1, 3 >

Program

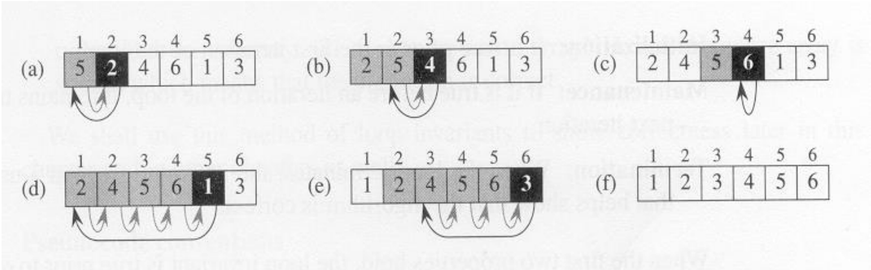
- Program by C

```
INSERTION-SORT(A)
for j ← 2 to length[A]
    do key ← A[j]
        ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
        i ← j − 1
        while i > 0 and A[i] > key
            do A[i + 1] ← A[i]
               i ← i − 1
        A[i + 1] ← key
```

```c
void InsertionSort(int* A)
{
    for(int j=1; j<LENGTH; j++)
    {
        int key = A[j];
        // Insert A[j] into the sorted sequence A[0..j-1]
        int i = j-1;
        while((i >= 0) && (A[i] > key))
        {
            A[i+1] = A[i];
            i--;
        }
        A[i+1]=key;
    }
}
```

Analysis

- Correctness
  - 알고리즘이 정확하게 동작함을 증명하는 방법
    - ➢ 연역적 방법 (deduction)
    - ➢ 귀납적 방법 (induction)



- Loop Invariant (루프 불변성)
  - 귀납적 방법

1) Initialization (초기조건)

   It is true prior to the 1st iteration of the loop : A[1]

2) Maintenance (유지조건)

   If it is true before an iteration of the loop, it remains true before the next iteration : A[1, … , j-1]

3) Termination (종료조건)

   when the loop terminates : A[1, … , n]

Analysis

- Running Time
  - *T(n)*
  - *n* : input size (*length*[A])

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

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

tj : *j* 에 대한 while loop 의 실행 횟수

$$t_j = \begin{cases} 1, & key(A[j]) \text{값이 왼쪽값들 중 최대일 때}, \textit{best case} \\ j, & key(A[j]) \text{값이 왼쪽값들 중 최소일 때}, \textit{worst case} \end{cases}$$

Analysis

- Running Time
  (1) Best Case

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$
$$= an + b$$

(ex) input = <1, 2, 3, 4, 5, 6>

  (2) Worst Case

$$T(n) = (\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2})n^2 + (c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8)n - (c_2 + c_4 + c_5 + c_8)$$
$$= an^2 + bn + c$$

(ex) input = <6, 5, 4, 3, 2, 1>

⇒ 일반적으로 running time은 Worst Case 를 기준으로 표현

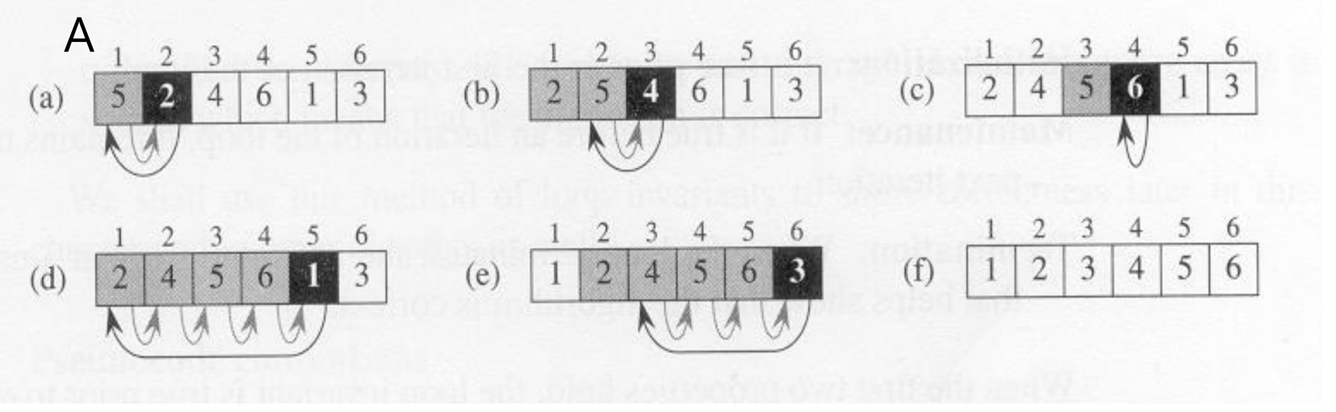
- Rate of Growth / Order of Growth
  - 충분히 큰 input size (n) 에 대한 running time 을 '근사적'으로 표현

$$an^2 + bn + c \quad \propto \quad an^2 \quad \propto \quad n^2$$

$$\boxed{\Theta(n^2)}$$

Analysis

- Memory
  - In-place sorting
    - 입력된 배열 (A) 내에서 정렬수행
    - 별도의 메모리를 사용하지 않음



Example

- Bubble Sort Algorithm

```
BUBBLESORT(A)
for i ← 1 to length[A]
    do for j ← length[A] downto i+1
        do if A[j] < A[j-1]
            Then exchange A[j] ↔ A[j-1]
```

1) A= < 5, 2, 4, 6, 1, 3>

2) Running Time (best / worst)

Example

- Linear search algorithm
  1) Problem

     input : A = <a_1, a_2, … , a_n> and a value v
     output : an index i such that v = A[i] or
     NIL if v does not appear in A
  2) Operation
     - 순차적으로 i 를 1부터 n 까지 증가시키며 탐색
     - incremental approach
  3) Algorithm

  4) Analysis

     - running time
     - memory