<u>CHAPTER 6</u>

SORTING

§1 Preliminaries

```
void  X_Sort ( ElementType  A[ ],  int N )

/* N must be a legal integer */

/* Assume integer array for the sake of simplicity */

/* '>' and '<' operators exist and are the only operations
   allowed on the input data */

/* Consider internal sorting only */
```

***Comparison-based* sorting**

The entire sort can be done in main memory

§2 Insertion Sort

```
void InsertionSort ( ElementType A[ ], int N )
{
    int  j, P;
    ElementType  Tmp;

    for ( P = 1; P < N; P++ ) {
        Tmp = A[ P ];  /* the next coming card */
        for ( j = P; j > 0 && A[ j - 1 ] > Tmp; j-- )
            A[ j ] = A[ j - 1 ];
            /* shift sorted cards to provide a position
               for the new coming card */
        A[ j ] = Tmp;  /* place the new card at the proper position */
    }  /* end for-P-loop */
}
```

The worst case: **Input A[ ] is in reverse order.** $T(N) = O(N^2)$

The best case: **Input A[ ] is in sorted order.** $T(N) = O(N)$

§3 A Lower Bound for Simple Sorting Algorithms

【Definition】 An inversion in an array of numbers is any ordered pair (*i*, *j*) having the property that *i* < *j* but A[*i*] > A[*j*].

〖Example〗 Input list 34, 8, 64, 51, 32, 21 has 9 inversions.

(34, 8) (34, 32) (34, 21) (64, 51) (64, 32) (64, 21) (51, 32) (51, 21) (32, 21)

There are 9 swaps needed to sort this list by insertion sort.

Swapping two adjacent elements that are out of place removes exactly one inversion.

$T(N, I) = O(I + N)$ where *I* is the number of inversions in the original array.

Fast if the list is almost sorted.

【Theorem】 **The average number of inversions in an array of N distinct numbers is $N (N - 1) / 4$.**

【Theorem】 **Any algorithm that sorts by exchanging adjacent elements requires $\Omega (N^2)$ time on average.**