Insertion sort

- solves the sorting problem

Input: A sequence of n numbers $\langle a_1, a_2, a_3 \cdots a_n \rangle$

Output: A permutation (reordering) of the input sequence

i.e. $\langle a_1', a_2' \cdots a_n' \rangle$ such that

$a_1' \leq a_2' \leq \cdots a_n'$

- it works the way many people sort a hand of playing cards.

- Analogy Description : Refer book

INSERTION SORT(A)

```
1.  for j ← 2 to length[A]
2.      do key ← A[j]
3.         ▷ Insert A[j] into the sorted sequence A[1···j-1]
4.         i ← j-1
5.         while i>0 and A[i] > key
6.             do A[i+1] ← A[i]
7.                i ← i-1
8.         A[i+1] ← key
```

Sorted in place

1) rearranged within the array and
2) at any time, at most a constant number of items are stored outside the array.

Design Method : - it uses an incremental design approach.

⇒ Having sorted subarray A[1···j-1], we insert the single element A[j] into its proper place to obtain the subarray A[1···j]

Loop invariants (LI)

j : current card being inserted into the hand.

At the beginning of each iteration of the outer for loop,

the subarray $A[1..j-1]$:- currently sorted hand

and $A[j+1..n]$:- pile of cards still on the table

⇒ In fact, elements $A[1..j-1]$ are the elements originally in positions 1 through $j-1$, but now in sorted order.

⇒ these properties of $A[1..j-1]$ may be used to define as a loop invariant

" At the start of each iteration of the for loop of lines 1-8, the subarray $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$ but in sorted order. "

- Loop invariant is used to understand the correctness of an algorithm

- we must show 3 things about a loop invariant:

i) Initialization: it is true prior to the 1st iteration of the loop.

ii) Maintenance: if it is true before an iteration of the loop, it remains true before the next iteration.

iii) Termination: when the loop terminates, the loop invariant gives us a useful property that helps to show that the algo is correct.

- when the first two properties hold, then the loop invariant is true prior to every iteration of the loop.

- note the similarity between L.I. and Mathematical Induction (M.I.)

M.I.	L.I.
base case	before the 1st iteration
inductive step	iteration to iteration
* inductive step is used infinitely	induction is stopped when loop terminates

- Let us see how these properties hold for insertion sort.

i) Initialization: (L.I. holds before 1st iteration) because when $j=2$, the subarray $A[1..j-1]$ consists of just single element $A[1]$, which is also originally at position $A[1]$. Moreover, this subarray is sorted trivially.

⇒ hence the L.I. holds prior to the 1st iteration of the loop.

(ii) Maintenance: (Each iteration maintains the L.I.)

the body of the outer for loop works by moving $A[j-1]$, $A[j-2]$, and so on by one position to the right until the proper position for $A[j]$ is found (lines 4-7), at which point the value of $A[j]$ is inserted (line 8).

⇒ i.e. 1st iteration is basically placing $A[2]$ at the right position so that $A[1]$ and $A[2]$ are in sorted order and originally at positions 1 & 2.

⇒ this means the loop body is maintaining the L.I.

iii) Termination : Finally, we examine what happens when loop terminates

Here, outer for loop ends when $j > n$ i.e. $j = n+1$

Now, we will use $j = n+1$ in the statement of L.I

So we have :-

"The subarray $A[1..n]$ consists of the elements originally in $A[1..n]$, but in sorted order."

⇒ But the subarray $A[1..n]$ is the entire array.

Hence, the entire array is sorted.

⇒ This implies that the algo is correct.