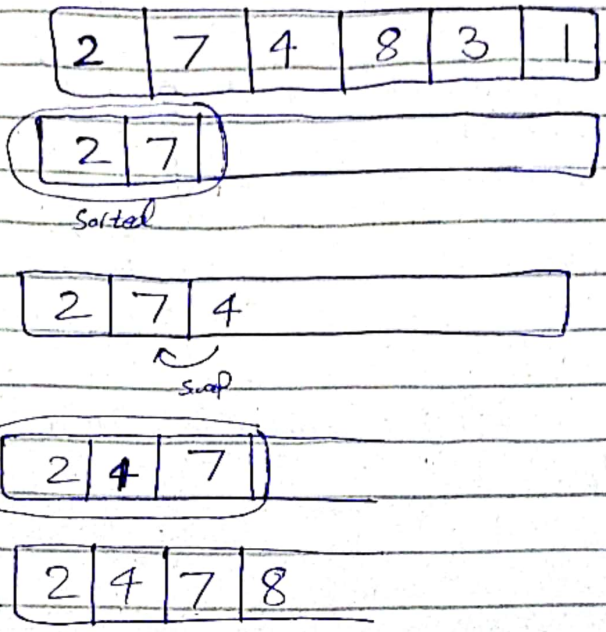
Insertion Sort:

∴ iteration number = ith element Sorted hain




```
insertion(A, n)        // n = size
{
    for(i ← 0 to n-1)
    {
        j ← i+1
        while(j > 0 && A[j] < A[j-1])
        {
            swapping(A[j], A[j-1])
            j--;
        }
    }
}
```

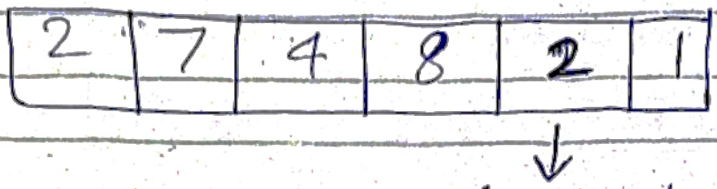
$$T(n) = 1+2+3+\cdots+n$$

$$T(n) = n^2$$

∴ This is inplace algorithm (using same array)

Bubble sort is also inplace algorithm

e.g.



This '2' should appear later in sorted array. This is called stable

→ If '2' swapped with 2 then this is unstable

Counting Inversions:

2	5	9	4	3	6

$0 \le i < j < n$

$\Downarrow$

$A[i] > A[j]$

then inversion exists

e.g. (5,4), (5,3) for 5

e.g. (9,4), (9,3), (9,6) for 6

∴ If less no. of inversions then time complexity should be less e.g. in case of insertion order $O(n)$ if we've less (1 or 2 etc) no. of inversions.

∴ Counting inversions should also be less.

Counting inversions in

sorted array = 0

unsorted array (descending) = maximum

e.g.

9	6	5	2	1
4in	3in	2	1	0

Total inversions $= 1+2+3+4$

$= 1+2+\cdots+(n-1)$

$= \frac{(n-1)[(n-1)+1]}{2}$

$= \frac{n(n-1)}{2} = {}^{n}C_2$

Counting Inversions' count:

Method 1:

```
CountInv(A,n)
{
  Count=0
  for(i<-0 to n-1)
    j=i+1
    while(j>0 && A[j]<A[j-1]
    {
      Count++
      j--
    }
  return Count;
}
```

$O(n^2)$

Method 2:

Divide & Conquer


```
Merge(A,l,mid,h)
{   count=0
  temp[h-l+1]      //new array
  i=l, j=mid+1, k=0

  while(i<=mid && ....)
  {
    if(A[i]<A[j])
      .....
    else
      .....
      Count = Count+(mid-i+1)
      :
  }
  }  return count
```

$O(n \log n)$