Input: { 1, 4, 7, 5, 2, 3 }

Output: { 1, 2, 3, 4, 5, 7 }

Inversions:
(4, 2)
(4, 3)
(7, 5)
(7, 2)
(7, 3)
(5, 2)
(5, 3)

Insertion Sort --

Initial : 1 4 7 5 2 3

Pass 1: 1 4 | 7 5 2 3 4>1

Pass 2: 1 4 7 | 5 2 3 7>4

Pass 3: 1 4 5 7 | 2 3 5<7, 5>4

Pass 4: 1 2 4 5 7 | 3 2<7, 2<5, 2<4, 2>1

Pass 5: 1 2 3 4 5 7 3<7, 3<5, 3<4, 3>2

Pass 4: j=3, j=2, j=1, j=0
Elements at indices (j+1) to (i-1) to be shifted

1 4 5 7 2 3
1 4 5 _ 7 3 tmp <- 2
1 4 _ 5 7 3 tmp <- 2
1 _ 4 5 7 3 tmp <- 2
1 2 4 5 7 3

```
InsertionSort( int[] A, int n )
  Input: An array A containing n >= 1 integers
  Output: The sorted version of the array A

for i = 1 to (n-1)
{
    j <- i - 1                          { 4, 1, ... }
    while A[i] < A[j]
          j <- j - 1
          if j < 0
              break
    tmp <- A[i]
    // shift all elements > A[i] by 1 position
    k = i - 1
    while k >= j+1
    {
          A[k+1] <- A[k]
          k <- k - 1
    }
    // insert A[i] in position (j+1)
    A[j+1] <- tmp
}
return A
```

Worst case complexity = O(n^2)
Best case complexity = O(n)

Input: { 7, 5, 4, 3, 2, 1 }

Output: { 1, 2, 3, 4, 5, 7 }

Insertion Sort --

Initial : 7 5 4 3 2 1

Pass 1: 5 7 | 4 3 2 1 5<7

Pass 2: 4 5 7 | 3 2 1 4<7, 4<5

Pass 3: 3 4 5 7 | 2 1 3<7, 3<5, 3<4

Pass 4: 2 3 4 5 7 | 1

Pass 5: 1 2 3 4 5 7

```
InsertionSortOptimized( int[] A, int n )
  Input: An array A containing n >= 1 integers
  Output: The sorted version of the array A

for i = 1 to (n-1)
{
    inversions = 0
    for j = 0 to (n-1)
        if A[j] > A[j+1]
              inversions <- inversions + 1
     if inversions == 0:
        break

     j <- i - 1
    while A[i] < A[j]
          j <- j - 1
          if j < 0
              break
    tmp <- A[i]
    // shift all elements > A[i] by 1 position
    k = i - 1
    while k >= j+1
    {
          A[k+1] <- A[k]
          k <- k - 1
    }
    // insert A[i] in position (j+1)
    A[j+1] <- tmp
}
return A
```

Worst case complexity = O(n^2)
Best case complexity = O(n)