Insertion Sort

```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 . . j − 1].
      i = j − 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = key
```

Loop invariant:

At the start of each iteration of the “outer” **for** loop – the loop indexed by j– the subarrary $A[1 \ldots, j-1]$ consists of the elements originally in $A[1, \ldots, j-1]$ but in sorted order.

Need to verify:

Similar to induction

- **Initialization:** It is true prior to the first iteration of the loop.
- **Maintenance:** If it is true before an iteration of the loop, it remains true before the next iteration.
- **Termination:** When the loop terminates, the invariant — usually along with the reason that the loop terminated — gives us a useful property that helps show that the algorithm is correct.

Insertion Sort

At the start of each iteration of the "outer" **for** loop – the loop indexed by j– the subarray $A[1 \ldots, j-1]$ consists of the elements originally in $A[1, \ldots, j-1]$ but in sorted order.

```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1..j-1].
      i = j - 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i - 1
      A[i + 1] = key
```

Initialization

- Before the first iteration of the loop we have $j = 2$.
- The subarray $A[1 \ldots j-1]$, therefore, consists of just the single element $A[1]$
- This is the original element in $A[1]$ and trivially sorted

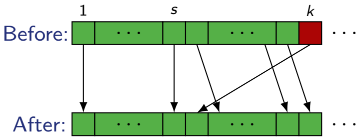
Insertion Sort

At the start of each iteration of the "outer" **for** loop – the loop indexed by j– the subarray $A[1 \ldots, j-1]$ consists of the elements originally in $A[1, \ldots, j-1]$ but in sorted order.

```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1..j-1].
      i = j - 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i - 1
      A[i + 1] = key
```

Maintenance:

- Assume invariant holds at the beginning of the iteration when $j = k$, i.e., for $A[1 \ldots k-1]$
- The body of the **for** loop works by moving $A[k-1], A[k-2]$ and so on one step to the right until it finds the proper position for $A[k]$, at which point it inserts the value of $A[k]$



s is largest index in $\{1, \ldots, k-1\}$ such that $A[s] \leq A[j] = A[k]$

or 0 if no such index exists

- The subarray $A[1 \ldots k]$ then consists of the elements originally in $A[1 \ldots k]$ in a sorted order. Incrementing j (to $k+1$) for the next iteration of the **for** loop then preserves the loop invariant :)

Insertion Sort

At the start of each iteration of the "outer" **for** loop – the loop indexed by j– the subarray $A[1 \ldots, j-1]$ consists of the elements originally in $A[1, \ldots, j-1]$ but in sorted order.

```
INSERTION-SORT(A, n)
  for j = 2 to n
      key = A[j]
      // Insert A[j] into the sorted sequence A[1 .. j - 1].
      i = j - 1
      while i > 0 and A[i] > key
          A[i + 1] = A[i]
          i = i - 1
      A[i + 1] = key
```

Termination

- The condition of the **for** loop to terminate is that $j > n$
- Hence, $j = n + 1$ when loop terminates
- The loop invariant then implies that $A[1 \ldots n]$ contain the original elements in sorted order