Straight Insertion Sort

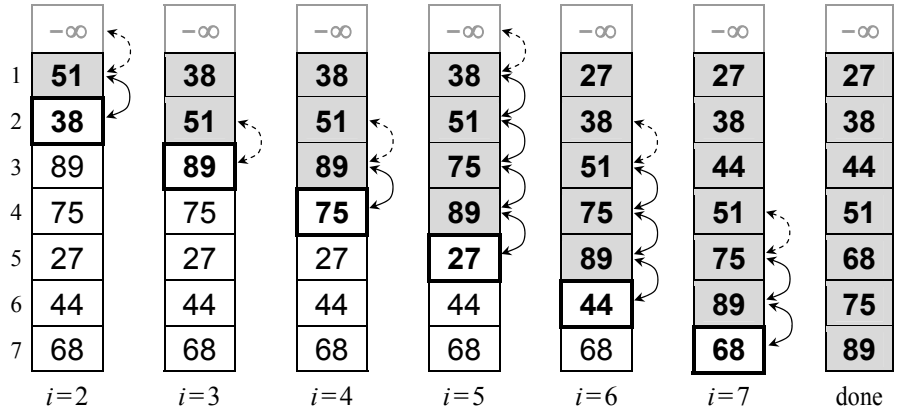
Shaded section of array contains original elements in this section, now rearranged in ascending order.

Boxed element is element to be inserted, so it becomes part of shaded area at the next step.

Unshaded unboxed section of array contains the original elements, completely untouched.

↻ = compare elements, and exchange them as they are out of order.

↻ = compare elements, and find them in order (no exchange).



Comparisons: $n-1$ comparisons in which the elements are in order.

I(A) comparisons in which the elements are out of order. (Each such comparison is followed by an exchange that removes one inversion.)

So $\boldsymbol{C(n) = I(A) + n - 1}$.

Exchanges: I(A)

$C_{max}(n) \approx n^2/2$, $C_{ave}(n) \approx n^2/4$, $C(n) << n^2/4$ if there is order present.

$T_{max}(n) = \Theta(n^2)$, $T_{ave}(n) = \Theta(n^2)$, $T(n)$ is much less if there is order present.

Input: An array A with element type T, and integers p and r with $lowerbound(A) \le p \le r \le upperbound(A)$.

Output: The array A, with $A[p..r]$ sorted, and any remaining elements of A unchanged.

Algorithm (implemented using exchanges)

```
void straight_insertion_sort( T[] A, Integer p, Integer r)
    for ( i = p+1, p+2, ..., r )          // Insert A[i] into already
        j = i;                            //  sorted subarray A[p..i–1].
        while ( j > p and A[j–1] > A[j] )
            swap( A[j–1], A[j]);
            j = j–1;
```

Algorithm (implemented using moves)

```
void straight_insertion_sort( T[] A, Integer p, Integer r)
    for ( i = p+1, p+2, ..., r )          // Insert A[i] into already
        temp = A[i];                      //  sorted subarray A[p..i–1].
        j = i;
        while ( j > p and A[j–1] > A[j] )
            A[j] = A[j–1];
            j = j–1;
        A[j] = temp;
```