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.

### Comparisons

\[ I(A) \text{ comparisons in which the elements are out of order.} \]

So \[ C(n) = I(A) + n - 1. \]

### Exchanges

\[ I(A) \]

---

**Algorithm (implemented using exchanges)**

```c
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)**

```c
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] > temp)
            A[j] = A[j–1];
            j = j–1;
        A[j] = temp;
}
```

---

**Input:** An array \( A \) with element type \( T \), and integers \( p \) and \( r \) with \( \text{lowerbound}(A) \leq p \leq r \leq \text{upperbound}(A) \).

**Output:** The array \( A \), with \( A[p..r] \) sorted, and any remaining elements of \( A \) unchanged.

**Comparsions:** \( n–1 \) comparisons in which the elements are in order.

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

So \[ C(n) = I(A) + n - 1. \]

**Exchanges:** \[ I(A) \]

---

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

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