### Insertion Sort: General Code Template

- An insertion sort code that works for every class
  - The `Comparable` interface imposes a total ordering on objects of a class
    - Any class that implements it must define a `compareTo` method
    - `x.compareTo(y)` returns a value less than, equal, or greater than 0

```java
public static <T extends Comparable<? super T>> void insertionSort(T[] a) {
    public static <T> void insertionSort(T[] a) {
        for (int p = 1; p < a.length; p++) {
            int tmp = a[p];
            int j = p;
            for (; ((j > 0) && tmp.compareTo(a[j - 1]) < 0); j--)
                for (; (j > 0) && tmp < a[j - 1]); j--
                    a[j] = a[j - 1];
            a[j] = tmp;
        }
    }
}
```

### In-class Problem

- **Posted on Piazza**
- How many positions are moved if insertion sort is used to completely sort the following data?
  - 3, 1, 4, 1, 5, 9, 2, 6, 5
- Is insertion sort a *stable* sorting algorithm?
- What’s the run-time for insertion sort?
  - Two nested for loops – O(N²)
**Recursive Formulation**

- **Recursive code for insertion sort:**
  - Recursively sort first $N - 1$ elements
  - Then insert $N^{th}$ element into sorted array

- **Run-time, $T(N)$: time to sort an array of size $N$**
  - $T(1) = c_1$
  - $T(N) \leq T(N-1) + c_2 N + c_3$

- **Solve recursive relation:**
  \[
  T(N) \leq T(N-1) + c_2 N + c_3 \\
  \leq (T(N-2) + c_4(N-1) + c_5) + c_2 N + c_3 \\
  \ldots \\
  \leq (c_2 N + c_4(N-1) + \ldots + c_3) \\
  \leq \max_i (c_i)^* \frac{N(N-1)}{2} \\
  = O(N^2)
  \]

**Insertion Sort: Properties**

- **Running time?**
  - Worst case is $O(N^2)$ – reversed input
  - Best case is $O(N)$ – for almost sorted data
    - What if all elements are equal?
  - Average case is $\Omega(N^2)$
    - True for any sorting algorithm that exchanges only adjacent elements

- **Space?**
  - Is an in-place sorting algorithm – $O(1)$ extra space

- **Sorting algorithm of choice when:**
  - Data is nearly sorted
  - Data is small (due to low overhead)
    - When input has shrunk in divide-and-conquer algorithms
A Lower Bound for Sorting

- **Average case run-time for insertion sort is** $\Omega(N^2)$
  - True for any sorting algorithm that exchanges only *adjacent* elements

- **Why?**
  - Inversion: any pair \((i, j)\), \(i < j\), such that \(a[i] > a[j]\)
  - Average number of inversions in an array: \(N(N-1)/4\)
    - Any list and its reverse have a total of \(N(N-1)/2\) inversions
  - In algorithms that swap only *adjacent* elements:
    - Each swap removes only one inversion
    - So, $\Omega(N^2)$ swaps are required

- **To do better, must:**
  - Compare and swap elements that are far apart
    - Try to eliminate more than just one inversion per swap