Sorting a List using an anonymous function

We assume you have read the one-page pdf file on sorting arrays using anonymous functions and understand it. We show you how to sort any List —for example, an `ArrayList` or even a `LinkedList`.

```
public class Person {
    public String first;
    public String last;
    public int age;
    …
}
```

Suppose we have declared this `LinkedList`:

```
List<Person> p= new LinkedList<>();
```

and have filled `p` with a bunch of `Person` objects. To sort `p` based on the ages of the Persons, use this statement:

```
p.sort((b, c) -> b.age - c.age);
```

To sort `p` in reverse order of age, use:

```
p.sort((b, c) -> c.age - b.age);
```

To sort `p` on the last name of the people in `p`, use

```
p.sort((b, c) -> b.last.compareTo(c.last));
```

Suppose we want to sort `p` by first name, but sort people with the same first name by age. Thus, if John Doe is age 9 and John Woe is age 5, put John Woe first.

Putting this all into an anonymous function is messy. Instead, we write static function `before` (look to the right). Then, it is a simple matter to write this call on sort:

```
p.sort((b, c) -> before(b, c));
```

```
/** Return a negative (positive) number if b's first
  * name comes before (or after) c's first name.
  * If b's and c's first names are the same,
  * return b's age - c's age. */
public static int before(Person b, Person c) {
    int n= b.first.compareTo(c.first);
    if (n != 0) return n;
    return b.age - c.age;
}
```

How does this work?

We read the documentation for interface `java.util.List`, procedure `sort`. It says that the default implementation of this procedure constructs an array containing all elements of the list, sorts the array, and then iterates over the list, resetting each element from the corresponding position in the array.

So it is simply doing the busy work that *you* would have to do if you wanted to sort a `LinkedList` and still have it take time O(n log n).

The documentation also says that the sort is a stable, adaptive, iterative mergesort. Also, it requires far fewer than $n\ lg(n)$ comparisons when the array of size n is partially sorted. Check the documentation for more information on how the sorting is done.