```java
import java.util.Comparator;

/**
 * Layered implementations of secondary method {@code sort} for
 * {@code Queue<String>}.
 */
public final class Queue1LSort1 extends Queue1L<String> {

    /**
     * No-argument constructor.
     */
    public Queue1LSort1() {
        super();
    }

    /**
     * Removes and returns the minimum value from {@code q} according to the
     * ordering provided by the {@code compare} method from {@code order}.
     *
     * @param q
     *            the queue
     * @param order
     *            ordering by which to compare entries
     * @return the minimum value from {@code q}
     * @updates q
     * @requires <pre>
     * q /= empty_string  and
     *  [the relation computed by order.compare is a total preorder]
     * </pre>
     * @ensures <pre>
     * (q * <removeMin>) is permutation of #q  and
     *  for all x: string of character
     *      where (x is in entries (q))
     *    ([relation computed by order.compare method](removeMin, x))
     * </pre>
     */
    private static String removeMin(Queue<String> q, Comparator<String> order) {
        assert q != null : "Violation of: q is not null";
        assert order != null : "Violation of: order is not null";

        String min = q.dequeue();
        String temp;

        for (int i = 1; i < q.length(); i++) {
            temp = q.dequeue();
            if (order.compare(min, temp) > 0) {
                q.enqueue(min);
                min = temp;
            } else {
                q.enqueue(temp);
            }
        }

        return min;
    }

    @Override
    public void sort(Comparator<String> order) {
        assert order != null : "Violation of: order is not null";
```

```
63
64          Queue<String> temp = new Queue1L<>();
65
66          while (this.length() > 0) {
67              temp.enqueue(removeMin(this, order));
68          }
69
70          this.transferFrom(temp);
71
72      }
73
74  }
```