```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();

        for (int i = 0; i < q.length(); i++) {
            String a = q.dequeue();

            if (order.compare(a, min) < 0) {
                q.enqueue(min);
                min = a;
            } else {
                q.enqueue(a);
            }

        }

        return min;
    }

```

```
61      @Override
62      public void sort(Comparator<String> order) {
63          assert order != null : "Violation of: order is not null";
64
65          Queue<String> temp = new Queue1L<>();
66
67          int length = this.length();
68          while (length > 0) {
69              temp.enqueue(removeMin(this, order));
70              length--;
71          }
72
73          this.transferFrom(temp);
74
75      }
76
77  }
```