```
    /**
     * 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>
     * perms(q * <removeMin>, #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)
{

        int index = 0;
        String min = q.dequeue();
        q.enqueue(min);;
        for(int i = 1; i < q.length(); i++) {
            String test = q.dequeue();
            if(order.compare(min, test) < 0) {
                min = test;
                index = i;
            }
            q.enqueue(test);
        }

        q.rotate(index + 1);

        return q.dequeue();
    }




/**
     * Sorts {@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 sort
 * @updates q
 * @requires [the relation computed by order.compare is a total preorder]
 * @ensures q = [#q ordered by the relation computed by order.compare]
 */
public static void sort(Queue<String> q, Comparator<String> order) {

    Queue<String> temp = q.newInstance();
    while(q.length() > 0) {
        temp.enqueue(removeMin(q, order));
    }
    q.transferFrom(temp);
}
```