```java
import java.util.Iterator;

/**
 * {@code Queue} represented as a {@code Sequence} of entries, with
 * implementations of primary methods.
 *
 * @param <T>
 *            type of {@code Queue} entries
 * @correspondence this = $this.entries
 */
public class HelloWorld {

    public static void main(String[] args) {
        Queue<Integer> test = new Queue1L<>();
        test.enqueue(1);
        test.enqueue(2);
        test.enqueue(3);
        test.rotate(1);
        System.out.println(test);

    }

    /**
     * 1 Returns the size of the given {@code Tree<T>}.
     *
     * @param <T>
     *            the type of the {@code Tree} node labels
     * @param t
     *            the {@code Tree} whose size to return
     * @return the size of the given {@code Tree}
     * @ensures size = |t|
     */
    public static <T> int size(Tree<T> t) {
        int count = 0;
        if (t.height() > 1) {
            count++;
            Sequence<Tree<T>> trees = new Sequence3<Tree<T>>();
            T root = t.disassemble(trees);
            for (Tree<T> tree : trees) {
                count += size(tree);
            }
            t.assemble(root, trees);

```

```
50          }
51
52          return count;
53      }
54
55      /**
56       * 2 Returns the size of the given {@code Tree<T>}.
57       *
58       * @param <T>
59       *            the type of the {@code Tree} node labels
60       * @param t
61       *            the {@code Tree} whose size to return
62       * @return the size of the given {@code Tree}
63       * @ensures size = |t|
64       */
65      public static <T> int size(Tree<T> t) {
66          int count = 0;
67          Iterator<T> iter = t.iterator();
68          while (iter.hasNext()) {
69              count++;
70          }
71
72          return count;
73      }
74
75      /**
76       * 3 Returns the height of the given {@code Tree<T>}.
77       *
78       * @param <T>
79       *            the type of the {@code Tree} node labels
80       * @param t
81       *            the {@code Tree} whose height to return
82       * @return the height of the given {@code Tree}
83       * @ensures height = ht(t)
84       */
85      public static <T> int height(Tree<T> t) {
86          int height = 0;
87          int maxHeight = 0;
88          if (t.size() > 0) {
89              Sequence<Tree<T>> trees = new Sequence3<Tree<T>>();
90              T root = t.disassemble(trees);
91              for (Tree<T> x : trees) {
92                  if (height(x) > maxHeight) {
93                      maxHeight = height(x);
```

```
 94                 }
 95             }
 96             height = maxHeight + 1;
 97             t.assemble(root, trees);
 98         }
 99     }
100 
101     /**
102      * 4 Returns the largest integer in the given {@code
    Tree<Integer>}.
103      *
104      * @param t
105      *            the {@code Tree<Integer>} whose largest integer
    to return
106      * @return the largest integer in the given {@code
    Tree<Integer>}
107      * @requires |t| > 0
108      * @ensures <pre>
109      * max is in labels(t)  and
110      * for all i: integer where (i is in labels(t)) (i <= max)
111      * </pre>
112      */
113     public static int max(Tree<Integer> t) {
114         int label = 0;
115         int maxLabel = 0;
116         if (t.size() > 0) {
117             Sequence<Tree<Integer>> trees = new
    Sequence3<Tree<Integer>>();
118             int root = t.disassemble(trees);
119             for (Tree<Integer> x : trees) {
120                 if (max(x) > maxLabel) {
121                     maxLabel = max(x);
122                 }
123             }
124             t.assemble(root, trees);
125         }
126     }
127 
128 }
```