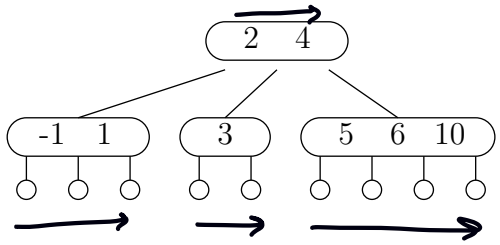
8. [7 points]

1:1 mapping
↳ if we have a choice, red on the left

a. Consider the following (2, 4) tree:

(2, 4) Tree



What is the sequence of keys encountered in the breadth-first traversal of this tree (going by layer and from left to right)? Choose one.

A ◯ `-1 1 3 5 6 2 4`

B ◯ `-1 1 2 3 4 5 6 10`

C ◯ `-1 1 3 2 5 6 10 4`

D ● `2 4 -1 1 3 5 6 10`

E ◯ `2 -1 1 4 3 5 6 10`

Continued

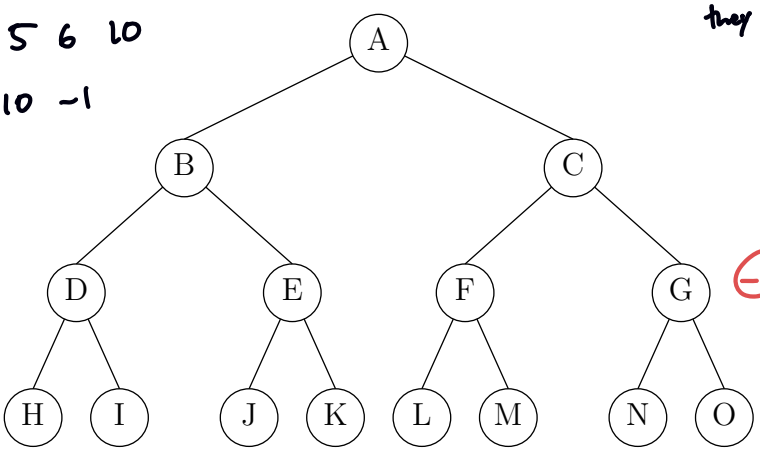
b. Fill in the unique left-leaning red-black tree corresponding to the (2, 4) tree in part (a). For each lettered node, indicate the key it contains in the space provided and if the node is red, fill in the corresponding bubble. Not all lettered nodes are actually needed. Leaving one blank indicates that it is actually an empty (null) node.

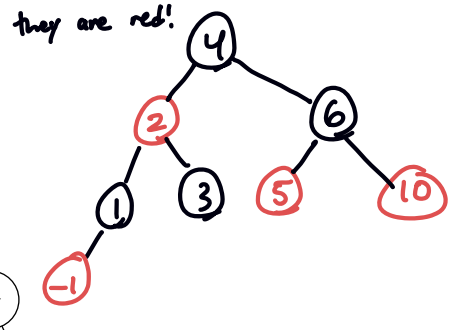
Left-Leaning Red-Black Tree

Want: 2 4 -1 1 3 5 6 10

Get: 4 2 6 1 3 5 10 -1

For some black node n: check its children to see if they are red!





A.	4	○ Red?	B.	2	● Red?	C.	6	○ Red?
D.	1	○ Red?	E.	3	○ Red?	F.	5	● Red?
G.	10	● Red?	H.	-1	● Red?	I.		○ Red?
J.		○ Red?	K.		○ Red?	L.		○ Red?
M.		○ Red?	N.		○ Red?	O.		○ Red?

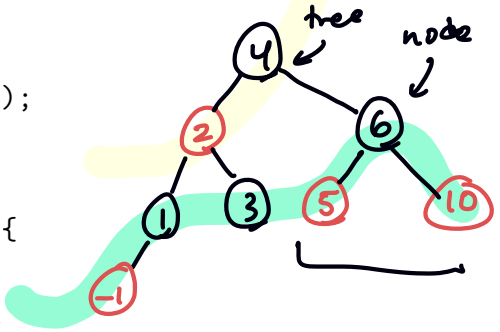
c. As you can see, the breadth-first traversal in part (a) does not correspond to a standard binary-tree breadth-first traversal (ignoring color) of the red-black tree of part (b). Fill in the implementation of the `rbBFS` method to match its specification, giving the same breadth-first traversal order for a left-leaning red-black tree as the breadth-first traversal of the corresponding (2, 4) tree.

Continued

Modified BFS

2 4 -1 1 3 5 6 10

Queue: ~~X~~ ~~X~~ ~~X~~ ~~X~~ (doesn't include null elements




```
1  public class RBT {
2      public static class RBNode {
3          int key;
4          RBNode left, right;
5          boolean isRed;
6      }
       /** Print the keys in TREE in the same order as for the (2, 4) tree
        *  corresponding to TREE. */
7      public static void rbBFS(RBNode tree) {
8          LinkedList<RBNode> queue = new LinkedList<>();
9          queue.add( tree );
10         while ( ! queue.isEmpty() ) {
11             RBNode node = queue.remove(0);
12             if ( node != null ) {
13 Checks →      if ( node.left != null && node.left.isRed ) {
14                 System.out.println( node.left.key ); print 2
15 left             queue.add( node.left.left ); add 1
16                 queue.add( node.left.right ); add 3
17             } else {
18                 queue.add(node.left); add 2
19             }
20 node      System.out.println( node.key );
21             if ( node.right != null && node.right.isRed ) {
22                 System.out.println( node.right.key );
23 right            queue.add(node.right.left);
24                 queue.add(node.right.right);
25             } else {
26                 queue.add(node.right);
27             }
28         }
29     }
30 }
31 }
```