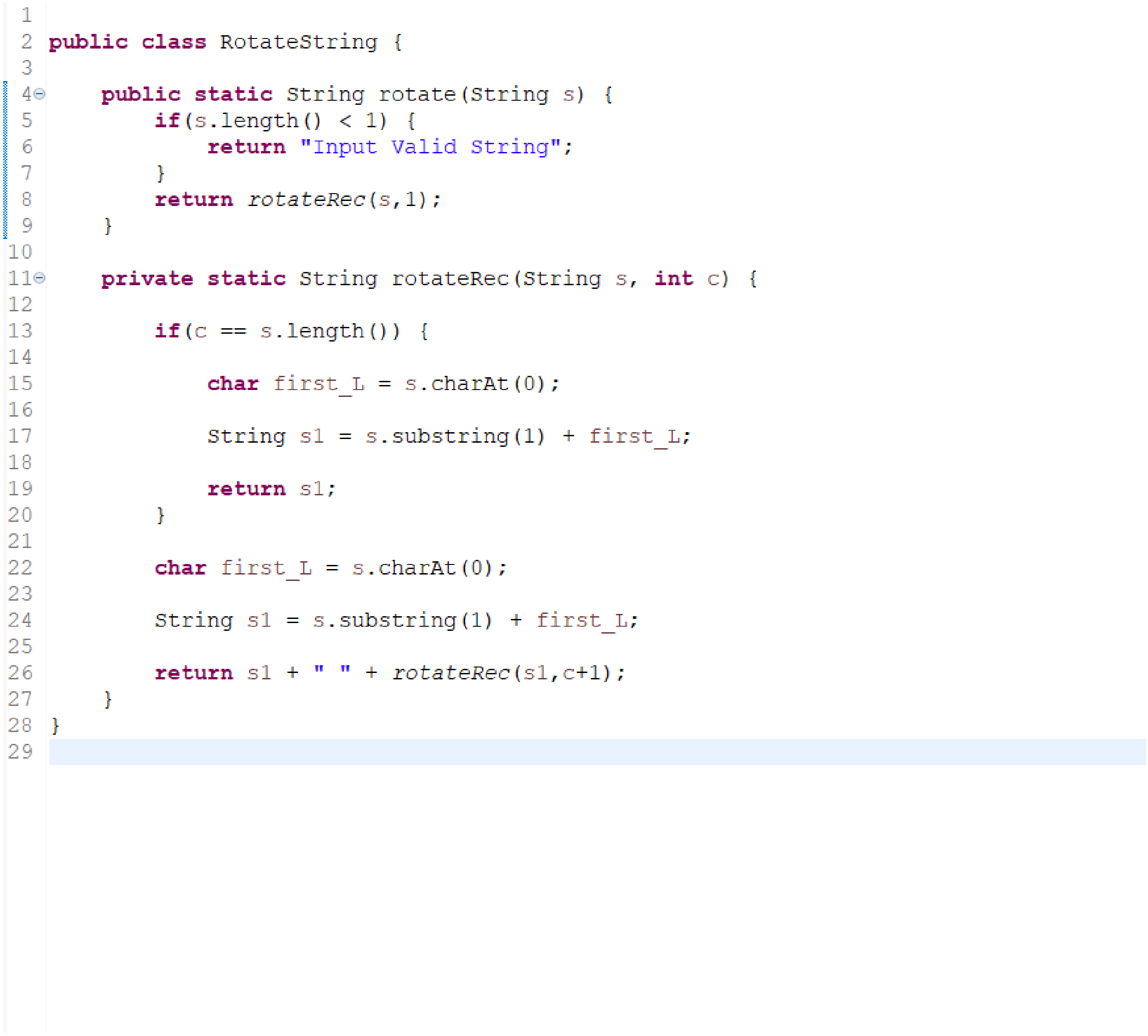
https://www.hackerrank.com/challenges/rotate-string/problem?h_r=internal-search

Pseudocode:

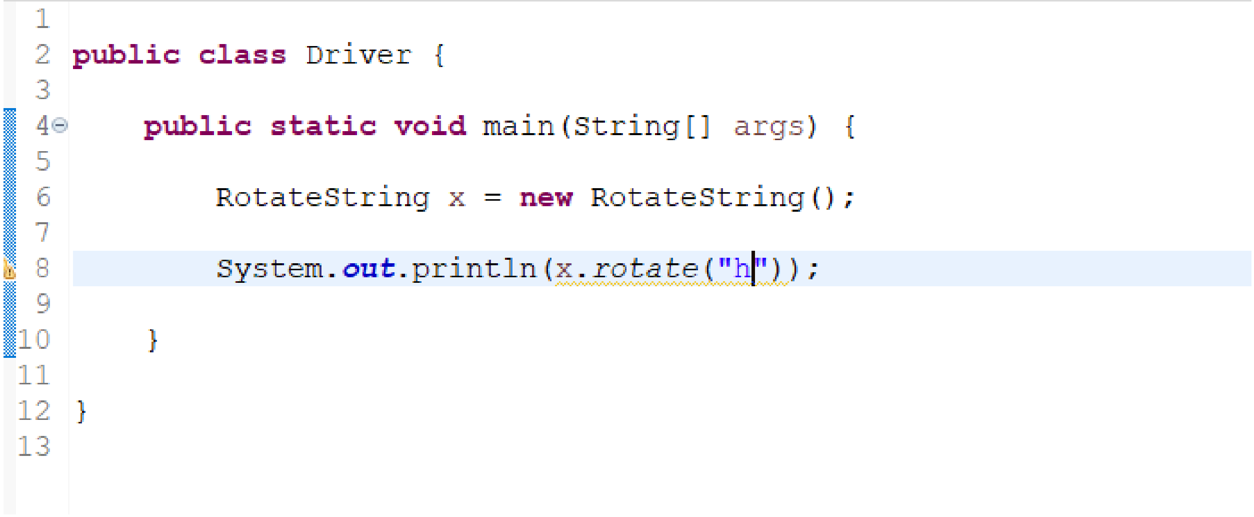
```
Rotate(String s):

if s.length < 1:
return “Invalid String”

return RotateRec(s,1)
```

```
RotateRec(String s , int c):

if s.length == c:
first_L = s[0]
s1 = s.substring(1:) + first_L
return s1

first_L = s[0]
s1 = s.substring(1:) + first_L
return s1 + “ ” + RotateRec(s1,c+1)
```

Java Implementation:



```java

public class RotateString {

    public static String rotate(String s) {
        if(s.length() < 1) {
            return "Input Valid String";
        }
        return rotateRec(s,1);
    }

    private static String rotateRec(String s, int c) {

        if(c == s.length()) {

            char first_L = s.charAt(0);

            String s1 = s.substring(1) + first_L;

            return s1;
        }

        char first_L = s.charAt(0);

        String s1 = s.substring(1) + first_L;

        return s1 + " " + rotateRec(s1,c+1);
    }
}

```

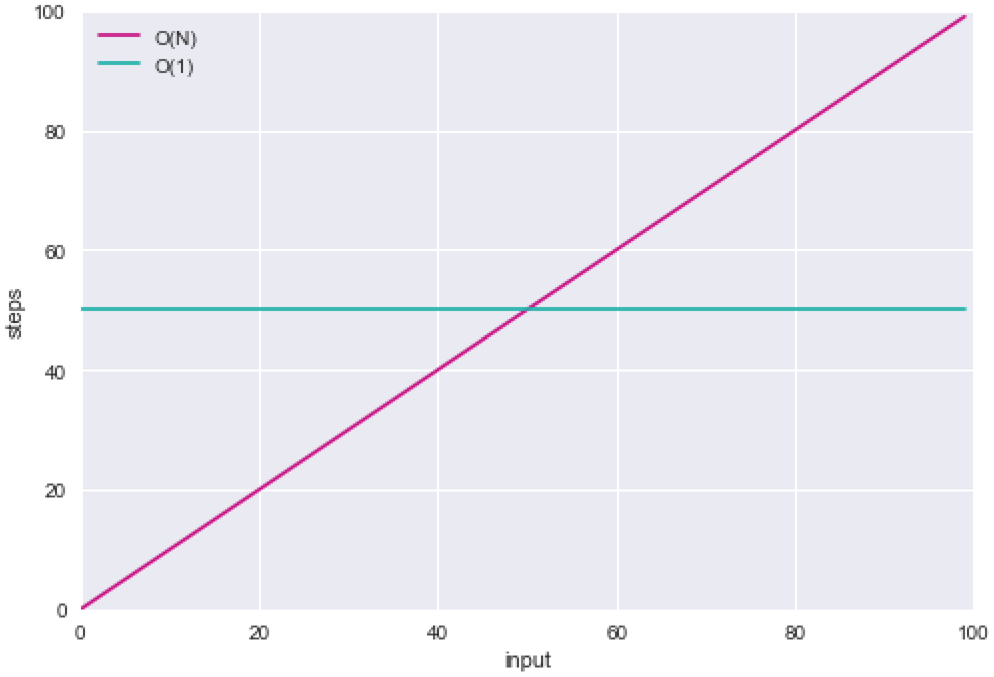


```java

public class Driver {

    public static void main(String[] args) {

        RotateString x = new RotateString();

        System.out.println(x.rotate("h"));

    }

}

```

While the above code uses a different Driver class to test the result, it still produces the desired output. Using recursion, the first character is isolated then added to the end of a substring not containing that same first character. The process is repeated until the string is fully rotated through, returning all the possible rotations separated by a space.

Time Complexity Analysis:

The time complexity of this program is **O(n).** This is because we recursively iterate over a String which in and of itself is a collection of characters similar to an array. This means that the time taken is only dependent on the length of string s, being n. So ultimately we can say that that time complexity of the program is O(n).


O(N)
O(1)
steps
input