LeetCode 33:
Find a target value in a rotated sorted array.

Brute force approach: perform linear search ==> Time complexity: O(n)

Binary search: allows us to find a target within a search pool of size n in O(log2(n))

Predicate: condition: find me an element == target
Predicate for the pivot: find me an element whose succesor is smaller

Time complexity: O(log2(n))

- Recursion:

Recursive method: Method that calls itself to fulfill its purpose. In order for this to be meaningful, the call to itself should be on a smaller version of the problem than the one originally attempted.

Example: Sum(N)
Sum of the values from 1 to N = N + Sum of the values from 1 to N-1 =
N + N - 1 + Sum of the values from 1 to N-2

Recursive methods have two main parts:
1. Manageable version of the problem: base case
N = 1 ===> Sum(1) = 1
2. Recursive part:
Sum(N) = N + Sum(N-1)

Example#2: SumRecApp

Iterative solution: Use repetition statements ===> Time complexity: O(n); Space complexity: O(1)
Recursive solution: ==> Time complexity: O(n); Space complexity: O(n) (recursive stack)

```
private static long sum_rec(int n) {
        // Base case
        if(n == 0) {
                return 0;
        }
        // Recursive part
        return n + sum_rec(n-1);
}
```

sum_rec(0) ===> return 0

```
sum_rec(1) ==> return 1 + 0 ---> 1
n = 0
sum_rec(0)

sum_rec(2) ==> return 2 + (sum_rec(1) ==> return 1 + sum_rec(0) (recursive path)

"Don't complain, just work harder"

- Example#3: FactorialRecApp
0! = 1
N! = N (N-1)!

BigInteger ===> Array of int values


- Example#4: Fibonacci
0       1       1       2       3       5 ...etc

golden ratio: 1.618

fib(n): Fibonacci function

Time complexity: O(2^n)
fib(0) = 0 and fib(1) = 1: base case
fib(n) = fib(n-1) + fib(n-2): recursive part
                        fib(4)

                fib(3)                  fib(2)
        fib(2)          fib(1)  fib(0)          fib(1)
fib(0)          fib(1)

n = 20 ==> 20 K recursive calls
n = 30 ==> 2 million recursive calls
n = 31 ==> 4 million recursive calls
n = 32 ==> 7 million recursive calls

Recursion + caching (memoization): Dynamic programming
Time complexity: O(n)
Space complexity: O(n)

Best solution for Fibonacci: Time complexity: O(log2(n))

0       1       1       2       ....etc
Time complexity: O(n)
Space complexity: O(n)


Dynamic programming: bottom-up approach
0, prevprev = 1,  prev=1, current = 2
Time complexity: O(n)
```

Space complexity: O(1)