Thinking in Recursion ¶

- a function that calls itself is a recusive function
- recursion uses stack memory
- recursive functions have notion of a `base` or `terminating` condition

1. Factorial
2. https://leetcode.com/problems/reverse-string/ (https://leetcode.com/problems/reverse-string/)
3. https://leetcode.com/problems/power-of-two/ (https://leetcode.com/problems/power-of-two/)
4. https://leetcode.com/problems/decode-string/ (https://leetcode.com/problems/decode-string/)
5. https://leetcode.com/problems/count-good-numbers/ (https://leetcode.com/problems/count-good-numbers/)

In [ ]:

Question
https://leetcode.com/problems/reverse-string/ (https://leetcode.com/problems/reverse-string/)

In [ ]:
```
class Solution {
    public void reverseString(char[] s) {

        // TC O(n) SC: O(n)
        reverseStringUtil(s, 0, s.length-1);


        // TC O(n) SC: O(1)
        // start = 0
        // end = s.length
        // while(start < end) {
        //     char temp = s[start];
        //     s[start] = s[end];
        //     s[end] = temp;
        //     start ++;
        //     end--;
        // }
    }

    public void reverseStringUtil(char[] s, int start, int end) {
        if (start >= end)
            return;

        char temp = s[start];
        s[start] = s[end];
        s[end] = temp;
        reverseStringUtil(s, start+1, end-1);
    }
}
```

In [ ]:
```
```

Question

https://leetcode.com/problems/power-of-two/ (https://leetcode.com/problems/power-of-two/)

In [ ]:
```
public class Solution {
    public boolean isPowerOfTwo(int n) {
        if (n <= 0) {
            return false;
        }
        if (n == 1) {
            return true;
        }
        if (n % 2 != 0) {
            return false;
        }
        return isPowerOfTwo(n / 2);
    }
}
```

In [ ]:
```
if n <= 0:
    return False
if n == 1:
    return True
while(n%2==0):
    n = n/2
return n==1
```

In [ ]:
```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if (n == 0) return false;
        while (n != 1) {
            if (n % 2 != 0) return false;
            n /= 2;
        }
        return true;
    }
}
```

In [ ]:
```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if (n==0)
            return false;
        if (n==1)
            return true;
        if(n%2==0){
            return isPowerOfTwo(n / 2);
        }
        else{
            return false;
        }
    }
}
```

In [ ]:
```
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n < 1)
            return false;
            if(n== 1)
            return true;
            if(n %2 ==1)
            return false;
            return (isPowerOfTwo(n/2));
    }
}
```

In [ ]:
```java
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n < 1)return false;
        if((n & n-1) == 0)return true;
        return false;
    }
}
```

In [ ]:
```java
public boolean isPowerOfTwo(int n) {
        if (n == 1) {
            return true;
        }
        if (n <= 0 || n % 2 != 0) {
            return false;
        }
        return isPowerOfTwo(n / 2);
    }
```

In [ ]:
```java
class Solution {
    public boolean isPowerOfTwo(int n) {
        if (n<1) return false;
        if (n==1) return true;
        if (n%2!=0) return false;
        return isPowerOfTwo(n/2);
    }
}
```

In [ ]:
```java
class Solution {
    public boolean isPowerOfTwo(int n) {

        if (n <= 0) {
            return false;
        }
        if (n == 1) {
            return true;
        }
        if (n % 2 != 0) {
            return false;
        }
        return isPowerOfTwo(n / 2);
    }
}
```

In [ ]:
```
4  = 00100  3 =  00011
8  = 01000  7 =  00111
5  = 00101  4 =  00100
24 = 11000 23 =  10111
```

Question

https://leetcode.com/problems/decode-string/ (https://leetcode.com/problems/decode-string/)

In [ ]:

In [ ]:

Question
https://leetcode.com/problems/count-good-numbers/ (https://leetcode.com/problems/count-good-numbers/)

In [ ]:

In [ ]:

In [1]:
```
def f1():
    print("f1")

f1()
```

```
f1
```

In [2]:
```
def f1():
    print("f1")

f1()
f1()
```

```
f1
f1
```

In [ ]:

In [3]:
```
def f1():
    print("f1")

def f2():
    print("f2")

f1()
```

```
f1
```

In [ ]:

In [4]:
```
def f1():
    print("f1")
    f2()

def f2():
    print("f2")

f1()
```

```
f1
f2
```

In [5]:
```
def f1():
    f2()
    print("f1")

def f2():
    print("f2")

f1()
```

```
f2
f1
```

In [6]:
```
def f1(n):
    f2(n-1)
    print(n)

def f2(n):
    print(n)

f1(10)
```

```
9
10
```

In [4]:
```
# Recursive function
def f1(n):
    if n == 0: # terminating condition
        return
    print(n)
    f1(n-1)

f1(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

In [5]:
```
# Recursive function
def f1(n):
    if n == 0: # terminating condition
        return
    f1(n-1)
    print(n)

f1(10)
```

```
1
2
3
4
5
6
7
8
9
10
```

Factorial of a number

n! = 123...(n-1)*n

```
long fact(int n) {
    long ans=1;
    for(int i = 1; i <= n; i++) {
        ans *= i;
    }
    return ans;
}

long factv2(int n) {
    if (n == 0) {
        return 1;
    }
    return n*factv2(n-1);
}

int main() {
    std::cout << "Hello World!\n" << factv2(5);
}
```

In [ ]:

In [ ]: