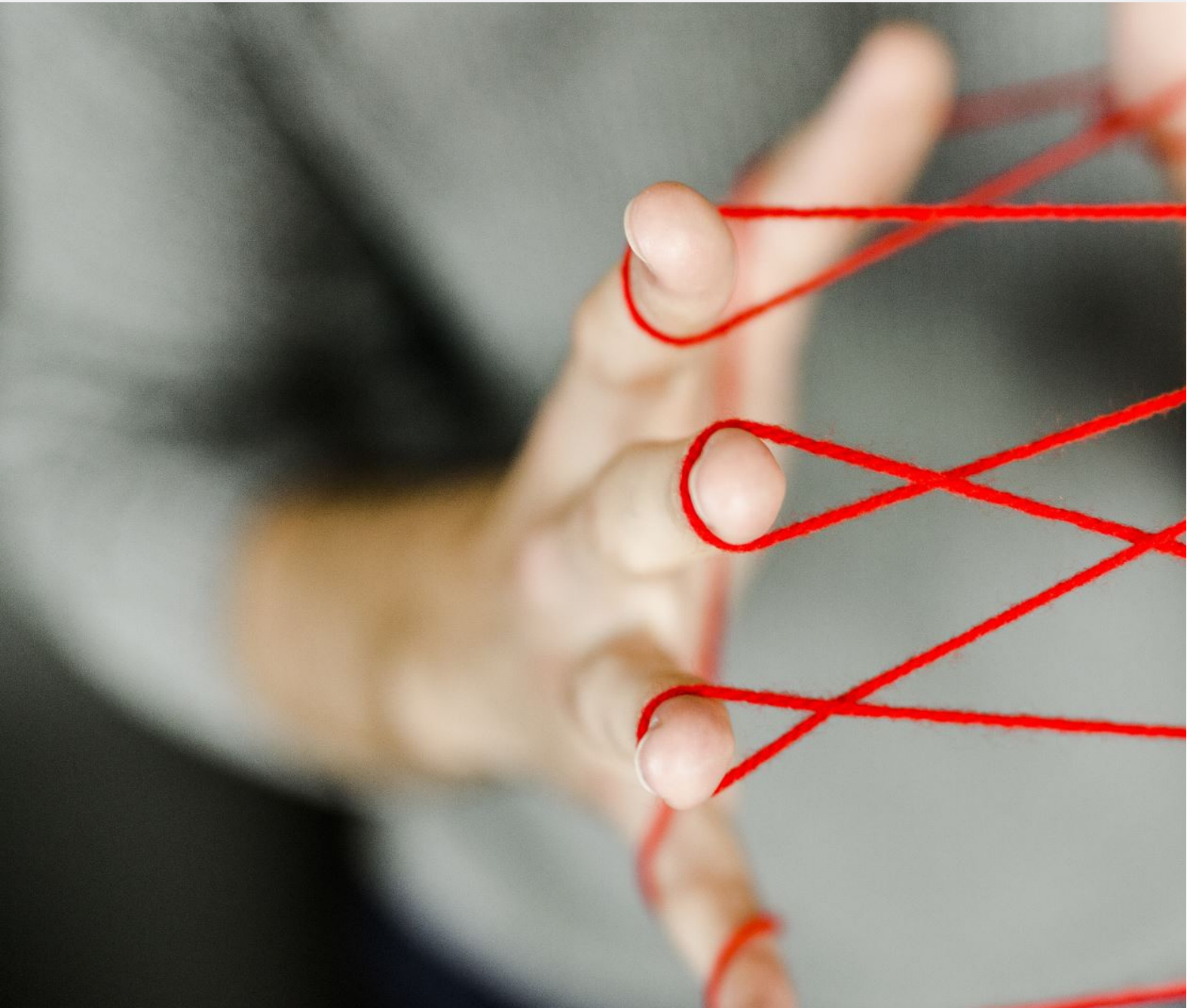
TIME TO WORK WITH RECURSION



CODING CHALLENGE

The Fibonacci sequence

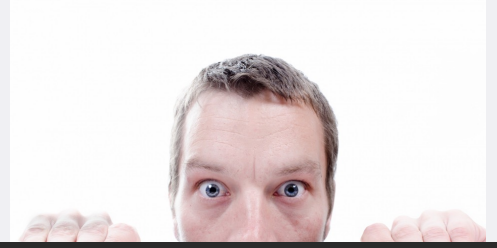
AS DISCUSSED EARLIER

```python
def fib(self, n):
    """
    Find the nth Fibonacci number using recursion
    :param n: the index of the Fibonacci number to find
    :return: the nth Fibonacci number
    """
    # Base case: if n is 0, the nth Fibonacci number is 0
    if n == 0:
        return 0
    # Base case: if n is 1, the nth Fibonacci number is 1
    elif n == 1:
        return 1
    # Recursive case: if n is greater than 1, the nth Fibonacci number is
    # the sum of the (n-1)th and (n-2)th Fibonacci numbers
    else:
        return self.fib(n - 1) + self.fib(n - 2)
```

$$T(n) = \begin{cases} \theta(1) & \text{if } n \leq 1 \\ T(n-1) + T(n-2) + \theta(1) & \text{otherwise} \end{cases}$$

$\theta(2^n)$

THE BAD FIB



$\Theta(2^n)$

```python
def bad_fib(self, n):
    """
    Find the nth Fibonacci number using recursion
    :param n: the index of the Fibonacci number to find
    :return: the nth Fibonacci number
    """
    # Base case: if n is 0, the nth Fibonacci number is 0
    if n == 0:
        return 0
    # Base case: if n is 1, the nth Fibonacci number is 1
    elif n == 1:
        return 1
    # Recursive case: if n is greater than 1, the nth Fibonacci number is
    # the sum of the (n-1)th and (n-2)th Fibonacci numbers
    else:
        return self.bad_fib(n - 1) + self.bad_fib(n - 2)
```

AS DISCUSSED EARLIER

```python
def fib(self, n):
    """
    Find the nth Fibonacci number using recursion
    :param n: the index of the Fibonacci number to find
    :return: the nth Fibonacci number
    """
    # Base case: if n is 0, the nth Fibonacci number is 0
    if n == 0:
        return 0
    # Base case: if n is 1, the nth Fibonacci number is 1
    elif n == 1:
        return 1
    # Recursive case: if n is greater than 1, the nth Fibonacci number is
    # the sum of the (n-1)th and (n-2)th Fibonacci numbers
    else:
        return self.fib(n - 1) + self.fib(n - 2)
```

$$T(n-1) + T(n-2) + \theta(1) \quad \text{otherwise}$$

Function calls itself twice, once with the input decremented by 1 and once with the input decremented by 2,
and performs a constant amount of non recursive work work

$$T(n) = \begin{cases} \theta(1) & \text{if } n \leq 1 \\ T(n-1) + T(n-2) + \theta(1) & \text{otherwise} \end{cases}$$

You could use Substitution or Recursion Tree.

$\theta(2^n)$

Master Theorem is not applicable for this recurrence.

Keep the recursion, use the memory

Bring the run time complexity to $\theta(n)$

HOW TO KEEP RECURSION BUT IMPROVE RUN TIME?

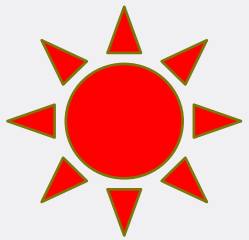
TAP INTO MEMORY

BAD FIBONACCI CAN BE IMPROVED

IF WE USE A DICTIONARY TO STORE RESULTS

This brings the run time from $\theta(2^n)$ to $\theta(n)$

GO TO WORK, COMPLETE GOOD_FIB



```python
    def good_fib(self, n: int) -> int:
        pass
        # Base case: if n is less than 2, the fibonacci number is n

        # Check if fib(n) has already been calculated and stored in memo

            # If it has, return the stored value


            # If it hasn't, calculate fib(n) by recursively finding fib(n-1) and fib(n-2)

        # Store the calculated value of fib(n) in memo for future use
```

Application problem that uses Recursion

Find the nested_list_product_sum

Expected run time complexity →O(n)

The most common recursion question

Definition of n in this problem is:

total number of elements , including the sub elements

PROBLEM DESCRIPTION

Implement a function that takes an “nested python list” and returns its product sum.

A nested list is a non-empty list that contains either integers, other nested lists.

The product sum of this nested list is the sum of its elements, where nested lists inside it are summed themselves, and then multiplied by their level of depth.

The depth of an interesting array is how far nested it is.

DEFINITION OF N IN THIS PROBLEM IS:

TOTAL NUMBER OF ELEMENTS , INCLUDING THE SUB ELEMENTS

PROBLEM DESCRIPTION

1. Single Depth Lists
   - `[2, 4]` -> 2 + 4 = 6
   - `[10, 15, 20]` -> 10 + 15 + 20 = 45
   - `[-1, 2, -3]` -> -1 + 2 - 3 = -2

2. Double Depth Lists
   - `[2, [4, 6]]` -> 2 + 2 * (4 + 6) = 22
   - `[10, [5], 15]` -> 10 + 2 * 5 + 15 = 35
   - `[-1, [2, -3], 4]` -> -1 + 2 * (2 - 3) + 4 = -1

3. Triple Depth Lists
   - `[2, [4, [6, 8]]]` -> 2 + 2 * (4 + 3 * (6 + 8)) = 74
   - `[10, [-5, [10]], 15]` -> 10 + 2 * (-5 + 3 * 10) + 15 = 80
   - `[-1, [2, [-3, 4]], 4]` -> -1 + 2 * (2 + 3 * (-3 + 4)) + 4 = 6

4. Mixed Depths
   - `[2, [4, 6], [8, 10]]` -> 2 + 2 * (4 + 6) + 2 * (8 + 10) = 48
   - `[10, [-5, [10, 15], 20], 25]` -> 10 + 2 * (-5 + 3 * (10 + 15) + 20) + 25 = 125

PROBLEM DESCRIPTION

For example :
The depth of [5] is 1, product sum is 5.
Depth of the inner array in [[ ]] is 2

The depth of the inner most array in [[[ ]]] is 3

The product sum of [3, 5] is 3 +5 → product sum is 8

Product sum of [3, [5,10]] is 3 + 2 * (5 + 10) → product sum is 33

The product sum of [3, [5, [10]]] is 3 + 2 * (5+ 3 *10) → product sum is 73

PROBLEM DESCRIPTION

The **nested_list_product_sum** function is a recursive algorithm that calculates the product sum of a nested list.
It takes an input array and a multiplier as parameters.

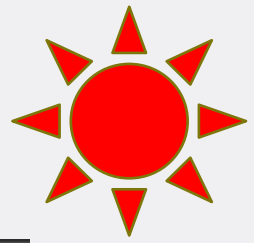
- The function iterates through the input array and for each element, it checks whether the element is a list or not.
    - If the element is a list, the function recursively calls itself with the element as the input array and the multiplier incremented by 1.
    - If the element is not a list, the function adds the element to the sum.

Finally, the function returns the sum multiplied by the multiplier.
The function returns the product sum of the input nested list.
The time complexity of the function is O(n) where n is the total number of elements in the input nested list.

LET'S GO TO WORK

DEFINITION OF N IN THIS PROBLEM IS:

TOTAL NUMBER OF ELEMENTS , INCLUDING THE SUB ELEMENTS



```
def nested_list_product_sum(self, data, depth=1):
    # Initialize sum to keep track of the total sum of all elements

    # Iterate through each element in the input array

        # Check if the current element is a list (nested list)

            # If it is a nested list, call the function recursively on that list
            # and add the returned value to the sum
            # Also, increment the depth by 1 for each level of nesting

        else:
            # If the element is not a list, add it to the sum

    # Return the final sum multiplied by the depth (to account for nested levels)
```

RUN TIME IS O(N)

Recall : Definition of n is n as total number of elements , including the sub elements

We defined n as the total number of elements, including the sub-elements, then the run time complexity of the nested_list_product_sum function would become O(n).

Explanation:

The function iterates through all the elements in the array, including the sub-elements in the nested lists, and performs a constant amount of work for each element.

Therefore, the total run time of the function is proportional to the number of elements in the array, which is O(n).

DEFINITION OF N IN THIS PROBLEM

n is total number of elements in the original array including all the elements in each sub array.

For example:

```
data5 = [5, 2, [7, -1], 3, [6, [-13, 8], 4]]
```

O(N) would be