## Recurrence Relation

**1. What is the time & space complexity of this algorithm?**

```python
# Q1
def mystery(N):
    if N == 1:
        return 1
    else:
        return mystery(N-1)
```

①

$$T(N) = \begin{cases} T(N-1) + a & ; \; N > 1 \\ b & ; \; N \end{cases}$$

b: constant

* identify which var is decreasing in each func call.
N becoming smaller ; smaller till we reach base case.

*Telescoping*

$T(N) = T(N-1) + a$

$= (T(N-2)+a) + a = T(N-2) + 2a$

$= (T(N-3)+a) + 2a = T(N-3) + 3a$

$\boxed{= T(N-k) + ka}$ general form

eliminate function call $(n=1)$
we want to reach base case.

Base when $T(1)$

$\therefore N - k = 1$

$k = N - 1$

when $k = N-1$

$T(N) = T(N-(N-1)) + (N-1)b$

$= T(N-N+1) + bN - b$

$= T(1) + bN - b$

$= a + bN - b$

$= O(N)$ #

**Time complexity: O(N)**
**Space complexity: O(N) = Input O(1) + Aux O(N) (Recursion height)**

2. What is the time & space complexity of this algorithm?

```python
# Q2
def mystery(N):
    if N == 1:
        return 1
    else:
        return mystery(N/2) + N
```

② 

$$T(N) = \begin{cases} T(N/2) + a & ; N > 1 \\ b & ; N = 1 \end{cases}$$

* $mys(N/2) + N$ — is a constant

Telescoping

$$T(N) = T(N/2) + a$$
$$= \left(T(N/4) + a\right) + a = T(N/2^2) + 2a$$
$$= \left(T(N/8) + a\right) + 2a = T(N/2^3) + 3a$$
$$\boxed{= T(N/2^k) + ka}$$

general form

Base when $T(1)$

$\therefore N/2^k = 1$

$N = 2^k$

$k = \log_2 N$

$$T(N) = T(N/2^{\log_2 N}) + (\log_2 N) b$$
$$= T(1) + \log_2 N\, b$$
$$= a + b \log_2 N$$
$$= O(\log N)$$

\#

**Time complexity: O(logN)**
**Space complexity: O(logN) = Input O(1) + Aux O(logN) (Recursion height)**

3. What is the time & space complexity of this algorithm?

```python
# Q3
def mystery(N):
    if N == 0:
        return 0
    else:
        total = 0
        for _ in range(3):
            total += mystery(N-1)
        return total
```

③

$$T(N) \begin{cases} 3T(N-1)+a & ; N>0 \\ b & ; N=0 \end{cases}$$

Base when $T(0)$

$\therefore N-k=0$

$k=N$

Telescoping

$$T(N) = 3T(N-1)+a$$

$$= 3(3T(N-2)+a)+a = 3^2T(N-2)+3a+a$$

$$= 3^2(3T(N-3)+a)+3a+a = 3^3T(N-3)+3^2a+3a+a$$

$$= 3^kT(N-k)+a(3^0+3^1+3^2+\dots 3^{k-1})$$

$$\frac{r^{n+1}-1}{r-1} \quad r=3, \; n=k-1 \quad = \frac{3^{k-1+1}-1}{3-1} = \frac{3^k-1}{2}$$

$$\boxed{= 3^kT(N-k)+a\left(\frac{3^k-1}{2}\right)}$$ general form

when $k=N$

$$T(N) = 3^NT(0)+a\left(\frac{3^N-1}{2}\right)$$

$$= 3^Nb+\tfrac{1}{2}a(3^N-1)$$

$$= O(3^N)$$ #

Time Complexity: O(3^N)
Space Complexity: O(N) = Input O(1) + Aux O(N) (Why? Because it will allocate and deallocate memory as it recurse and returns.)

4. What is the time & space complexity of this algorithm?

```python
# Q4
def mystery(N):
    if N == 0:
        return 0
    else:
        value = 0
        for i in range(N):
            value += i
        return value + mystery(N-1)
```

④

$$T(N) \begin{cases} N\,T(N-1) + Nb + c \quad \text{OR} \quad T(N-1) + Nb + c \ ; & N > 0 \\ a \ ; & N = 0 \end{cases}$$

be careful! function calling not in for loop.

Telescoping

$$T(N) = T(N-1) + Nb + c$$

$$= \left(T(N-2) + (N-1)b + c\right) + Nb + c = T(N-2) + (N-1)b + Nb + 2c$$

$$= \left(T(N-3) + (N-2)b + c\right) + (N-1)b + Nb + 2c = T(N-3) + Nb + (N-1)b + (N-2)b + 3c$$

$$= T(N-k) + \left(\sum_{i=n-k+1}^{n} i\right) b + kc$$

$$\left(N + (N-1) + (N-2) + \ldots + (N-k+1)\right) b$$

$$T(N-k) + \left( \sum_{i=n-k+1}^{n} i \right) b + kc$$

Base when $T(0)$

$$\therefore N - k = 0$$

$$k = N$$

when $k = N$

$$T(N) = T(N-N) + \left( \sum_{i=n-n+1}^{n} i \right) b + Nc$$

$$= T(0) + (1 + 2 + \dots + N) b + Nc$$

$$= a + \left( \frac{n(n+1)}{2} \right) b + Nc$$

$$= O(N^2) \#$$

Time Complexity: O(N^2)
Space Complexity: O(N) = Input O(1) + Aux O(N) (Why? Because the loop doesn't do anything for the space complexity).

5. Identify the space & auxiliary complexity of this algorithm

```python
# Q5
def binarySearch(array, left, right, key):
    if right >= left:
        mid = left + (right - left) // 2

        if array[mid] == key:
            return mid
        elif array[mid] > key:
            return binarySearch(array, left, mid-1, key)
        else:
            return binarySearch(array, mid+1, right, key)

    return -1
```

Auxiliary Space : The extra or temporary space used by an algorithm. (In other words, space that doesn't account for a function's/algorithm's input size).

The above is recursive implementation of binary search:
Auxiliary space: O(logN) (recursive height)
Space complexity: O(N)

If it's iterative implementation of binary search (Ian's lecture video):
Auxiliary Space: O(1) (in-place)
Space complexity: O(N)

As you can see here, comparing the space used by both types of function (recurrence and iterative), although both of their space complexity is O(N). But with auxiliary, we can identify with ease the amount of additional space used by the functions.