WELCOME TO

FIT 2 4’s

PASS session

How to score well in this unit

- Keep up to date with the lecture contents(don't let it snowball)

- Attend sanity check (IMPORTANT)

- Watch all tutorial videos(including DP)

- Revise past content every so often to avoid forgetting everything

- Be a tryhard and start assignment early

RECURRENCE RELATION

A)

```
def function_one(n):
    if n == 0:
        return 0
    else:
        return function_one(n-1)
```

1) **Closed Form:**

$T(0) = a$

$T(n) = T(n-1) + c \quad ; n > 0$

2) **Perform telescoping:**

$T(n) = T(n-1) + c$

$= T(n-2) + c + c$

$= T(n-3) + c + c + c$

$= T(n-3) + 3c$

3) **Solve for general form:**

$T(n) = T(n-k) + kc$

4) **Identify when will base case occur:**

Base when $T(0)$:

$n-k = 0$

$n = k$

5) **Substitute base case in:**

When $n = k$:

$T(n) = T(n-n) + nc$

$= T(0) + nc$

$= a + nc$

$= O(n)$

B)

```
def function_two(n):
   if n == 1:
       return 0
   else:
       return function__two(n//2) + 10
```

1) **Closed Form:**

$T(1) = a$

$T(n) = T(\frac{n}{2}) + c \quad ; n > 1$

2) **Perform telescoping:**

$T(n) = T(\frac{n}{2}) + c$

$= [T(\frac{n}{4}) + c] + c$

$= T(\frac{n}{4}) + c + c$

$= [T(\frac{n}{8}) + c] + c + c$

$= T(\frac{n}{8}) + 3c$

3) **Solve for general form:**

$T(n) = T(\frac{n}{2^k}) + kc$

4) **Identify when will base case occur:**

Base when $T(1)$:

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

5) **Substitute base case in:**

When $k = \log_2 n$:

$T(n) = T(\frac{n}{2^{\log_2 n}}) + \log_2 nc$

$= T(\frac{n}{n}) + \log_2 nc$

$= T(1) + \log_2 nc$

$= a + \log_2 nc$

$= O(\log n)$

C)

```
def function_three(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    else:        every time call itself two time like Fib
        return function_three(n-1) +
               function_three(n-2)
```

1) Closed Form:

T(0) = a
T(1) = b
$T(n) < 2T(n-1) + c \quad ; n > 1$

2) Perform telescoping:

$T(n) < 2T(n-1) + c$
$< 2[2T(n-2)+c] + c$
$< 2^2 T(n-2) + 2c + c$
$< 2^2 [2T(n-3) + c] + 2c + c$
$< 2^3 T(n-3) + 4c + 2c + c$
$< 2^3 T(n-3) + 2^2c + 2^1c + 2^0c$
$< 2^3 T(n-3) + c(2^0+2^1+2^2\ldots)$

3) Solve for general form:

$T(n) < 2^3 T(n-3) + 2^2c + 2^1c + 2^0c$
$< 2^3 T(n-3) + c(2^0+2^1+2^2\ldots)$
$< 2^k T(n-k) + (2^k-1)c$

4) Identify when will base case occur:

Base when T(0):
$n - k = 0$
$k = n$

5) Substitute base case in:

$T(n) < 2^n T(n-n) + (2^n-1)c$
$< 2^n T(0) + (2^n-1)c$
$< 2^n a + 2^n c - c$
$\approx O(2^n)$

D)

```
def function_four(n, m):
    if n == 0:
        return m   output, complexity count how many
    else:          time need to call it self
        return 3 * function_four(n//3, m)
```

Ian also explains this when he solved question 6 tut 2

1) Closed Form:

T(0) = a
T(1) = b   <- extra base case
$T(n) = T(\frac{n}{3}) + c \quad ; n > 1$

2) Perform telescoping:

$T(n) = T(\frac{n}{3}) + c$

$= [T(\frac{n}{9}) + c] + c$

$= T(\frac{n}{9}) + 2c$

$= [T(\frac{n}{27}) + c] + 2c$

$= T(\frac{n}{27}) + 3c$

3) Solve for general form:

$T(n) = T(\frac{n}{3^k}) + kc$

4) Identify when will base case occur:

$\frac{n}{3k} = 1$
$n = 3^k$
$k = \log_3 n$

5) Substitute base case in:

$T(n) = T(\frac{n}{3^{\log_3 n}}) + \log_3 nc$

$= T(\frac{n}{n}) + \log_3 nc$

$= T(1) + \log_3 nc$

$= b + \log_3 nc$

$= O(\log n)$

For constants *b* and *c*, consider the recurrence relation given by:

- T(n) = b, if n=1
- T(n) = 2 * T(n/2) + c * n^3, if n>1

Which of the following statements is true?

Select one:

- a. T(n) = $\Theta(n^3 * \log n)$
- b. T(n) = $\Theta(n^4)$
- c. T(n) = $\Theta(n^3)$
- d. T(n) = $\Theta(n^6 * \log n)$
- e. T(n) = $\Theta(n^3 * \log n * \log n * \log n)$

*2022 Semester 1 Q1 (2 marks)

1) Perform telescoping:

$$T(n) = 2T(\frac{n}{2}) + cn^3$$

$$= 2[2T(\frac{n}{4}) + c(\frac{n}{2})^3] + cn^3$$

$$= 2^2 T(\frac{n}{4}) + 2c(\frac{n}{2})^3 + cn^3$$

$$= 2^2 [2T(\frac{n}{8}) + c(\frac{n}{4})^3] + 2c(\frac{n}{2})^3 + cn^3$$

$$= 2^3 T(\frac{n}{8}) + 4c(\frac{n}{4})^3 + 2c(\frac{n}{2})^3 + cn^3$$

$$= 2^3 T(\frac{n}{8}) + \frac{4cn^3}{64} + \frac{2cn^3}{8} + cn^3$$

2) Solve for general form:

$$T(n) = 2^k T(\frac{n}{2^k}) + cn^3(1 + \frac{1}{4} + \frac{1}{16} + \ldots + \frac{1}{4^{k-1}})$$ = (1/4)^(k-1)

r = 1/4 k = k -1

$$= 2^k T(\frac{n}{2^k}) + cn^3(\frac{\frac{1}{4}^k - 1}{\frac{1}{4} - 1})$$ <- use the identity of problem 3 given in Tutorial 1

$$= 2^k T(\frac{n}{2^k}) + cn^3 (\frac{2^{-2k} - 1}{-\frac{3}{4}})$$

$$= 2^k T(\frac{n}{2^k}) - \frac{4}{3} cn^3(2^{-2k} - 1)$$

3) Identify when will base case occur:

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

4) Substitute base case in:

$$T(n) = 2^{log_2 n}\ T(\frac{n}{2^{log_2 n}}) - \frac{4}{3}\ cn^3(2^{-2log_2 n} - 1)$$

$$= n\ T(\frac{n}{n}) - \frac{4}{3}\ cn^3((2^{log_2 n})^{-2} - 1)$$

$$= n\ T(\frac{n}{n}) - \frac{4}{3}\ cn^3(n^{-2} - 1)$$

$$= nb - \frac{4}{3}\ cn^3(n^{-2} - 1)$$

$$= nb - \frac{4}{3}\ cn + \frac{4}{3}\ cn^3$$

$$= O(n^3)$$

AUXILIARY SPACE COMPLEXITY

- **In place algorithm is an algorithm with O(1) auxiliary space, NOT no auxiliary space**

```python
def aux_one(n):
    arr = [None] * n
    for i in range(n):
        arr[i] = i * 2
    return sum(arr)
```

- Input given is O(1) space
- In line 2, an array of size n is created, hence O(n) auxiliary space complexity so far
- For loop only modifies value, not memory
- Hence, aux space comp O(n), overall space comp O(n)

```python
def aux_two(arr):
    for i in range(len(arr)):
        arr[i] += 1
    return arr
```

- Input given is O(n) space
- For loop only modifies value, not memory
- Hence, aux space comp O(1), overall space comp O(n)

```python
def aux_four(n):
    matrix = [None] * n
    for i in range(n):
        matrix[i] = [None] * n
    return 1000
```

- Input given is O(1) space
- In line 2, an array of size n is created, hence O(n) auxiliary space complexity so far
- For each element in matrix, an additional array is created of size n. Hence, O(n) auxiliary space comp inside each element of the array
- Hence, aux space comp $O(n^2)$, overall space comp $O(n^2)$

1)

What is the **auxiliary space complexity** of the following algorithm (expressed in big-O)?

The input *n* is **always a positive integer**.

```
f(n):
  lst = [None]*n
  for i in range(n):
    lst = [i] * n   = [i, i, i] for n =3
```

*2021 Semester 1 Q7 (1 mark)

Answer is O(n).

Firstly, an array of size n is created. When entering the for loop, at each iteration, a new array is created, but it replaces the array created in the previous iteration, hence it is not $O(n^2)$.

Good way to visualize this is to use Python tutor. I have dropped the link below.

2)

Given an algorithm which runs in *O(NlogN)* time, which of the following are **possible auxiliary space complexities** for this algorithm?

Mark all that are possible.

Select one or more:

☐ a. O(1)

☐ b. O(N)

☐ c. O(NlogN)

☐ d. O(N^2)

*2021 Semester 1 Q6 (1 mark)

Answer is all of them except for d.

Time comp >= Aux space comp

To elaborate on this further, it goes back to what you understand in MIPS from FIT1008. When we want to create an array of size n, we have to manually allocate n memory. Hence, if we have O(n^2) aux space, we have to manually allocate O(n^2) memory. This is why your time complexity is bounded by your auxiliary space complexity.

https://pythontutor.com/visualize.html#mode=display

Just dropping the link for Python tutor as it is quite good for visualizing auxiliary space in case you get confused.