Your Ultimate Guide To Landing Top AI roles

2.3.6

Time Complexity of Recursive Programs

Ex 1 : Sum of first n natural number.

```
def summ(n):
    if n==0:
        return 0
    return n+summ(n-1)
```

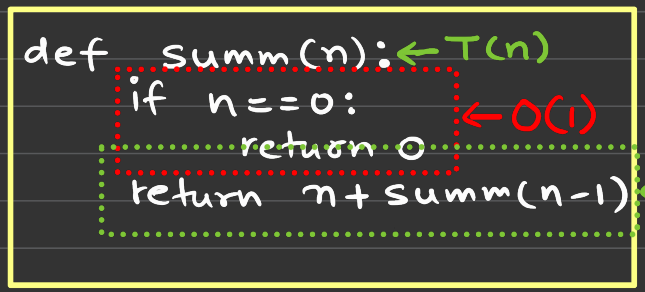
① we will use Big-O notation to find time Complexity.

Big-O : Worst Case time Complexity

② To find Time Complexity of Recursive programs, we write a **recurrence relation** and then try to analyze them.

Recurrence relation is a specific type of recursive equation that expresses time (or space) Complexity in terms of input size

→ Let's analyze the time Complexity of above Recursive Programs.



→ Step 1 : Recurrence relation.

T(n) = Time Complexity of summ(n) function when input size is 'n'.

$$T(n) = T(n-1) + \underbrace{O(1)}_{c}$$
$$T(0) = O(1)$$

↑ Recurrence relation.

☆ How to Solve any Recurrence relation?

① Back Substitution

② Recursion Tree

③ Master's Theorem.

→ Let's solve the Recurrence relation

$$T(n) = T(n-1) + \underbrace{O(1)}_{c}$$

$$T(0) = \underbrace{O(1)}_{c}$$

① Back Substitution.

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

$T(n-1) = T(n-2) + c$

$T(n-2) = T(n-3) + c$

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

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

$$= c + c + c + c + T(n-4) = 4c + T(n-4)$$

$$\vdots$$

$n - k = 0$

$k = n$

$$= k \cdot c + T(n-k)$$

$$= n \cdot c + T(0)$$

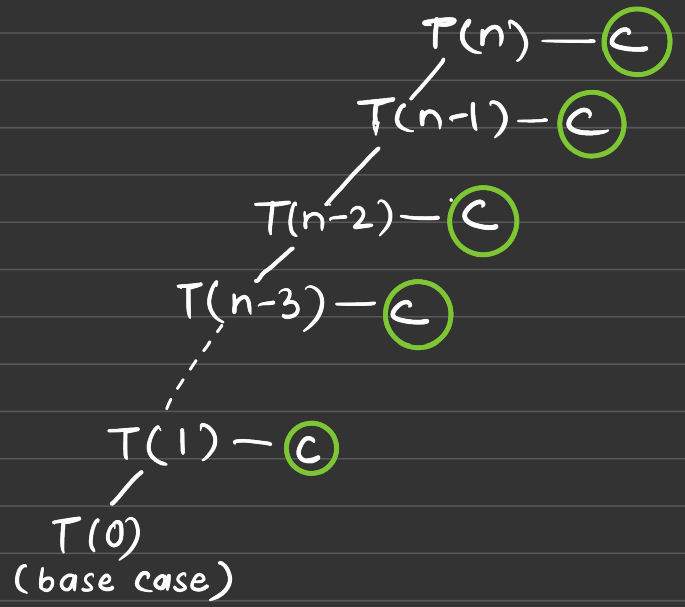
$$= n \cdot c + c$$

$$\boxed{T(n) = O(n)}$$

② Recursion Tree

$$T(n) = T(n-1) + O(1), \quad n > 0$$

$$T(0) = 1$$



height = n

Total time = n x c
= nc unit time

$T(n) = nc$

$\boxed{T(n) = O(n)}$

③ Master's Theorem
↳ Master's theorem docesn't fit for above equation

④ Intuition
↳ based on problem Solving

Master's Theorem

→ General form of equation for master's Theorem.

$$T(n) = a\,T(n/b) + f(n)$$

where $f(n) = \theta(n^k \log^p n)$

and $a \geq 1, b > 1, k \geq 0$ and p is real no.

① Case 1: $a > b^k$

$$T(n) = \theta(n^{\log_b a})$$

② Case 2: $a = b^k$

↳ 3 case based on $p > -1$, $p = -1$, $p < -1$

$$T(n) = \theta(n^{\log_b a} \cdot \log^{p+1} n) \quad \text{if } p > -1$$

$$= \theta(n^{\log_b a} \cdot \log\log n) \quad \text{if } p = -1$$

$$= \theta(n^{\log_b a}) \quad \text{if } p < -1$$

In master's theorem $T(n)$ is in θ instead of O.
So $T(n)$ is tighter upper bound

③ Case 3 : $a < b^k$

$$T(n) = \theta(n^k \cdot \log^p n) \quad \text{if } p \geq 0$$

$$= \theta(n^k) \quad \text{if } p < 0$$

Problem 2 : Merge Sort

```
def merge_sort(arr, lidx, ridx)          ← T(n)
    if len(arr) <= 1:                    } c unit
        return arr

    mid = (ridx - lidx)/2                } c unit
    merge_sort(arr, lidx, mid)           ← T(n/2)
    merge_sort(arr, mid+1, ridx)         ← T(n/2)
    merge(arr, lidx, ridx)               ← O(n)
    return
```

① Recurrence relation

$$T(n) = 2 \cdot T(n/2) + \underbrace{O(n) + 2 \cdot c}_{n \cdot c + 2 \cdot c \; O(n)}$$

$$T(n) = 2T(n/2) + O(n) \quad \} -①$$

$$= O(1), \; n <= 1$$

② Solving Recurrence relation.

→ Master's Theorem. $T(n) = a \cdot T(n/b) + \theta(n^k \log^p n)$ — ②

from eqn ① and ②

$a = 2, b = 2, k = 1, p = 0$

Here $a = b^k$ $(2 = 2^1)$ — Case ②

Also $p > -1 \Rightarrow T(n) = \theta(n^{\log_b^a} \cdot \log^{p+1} n)$

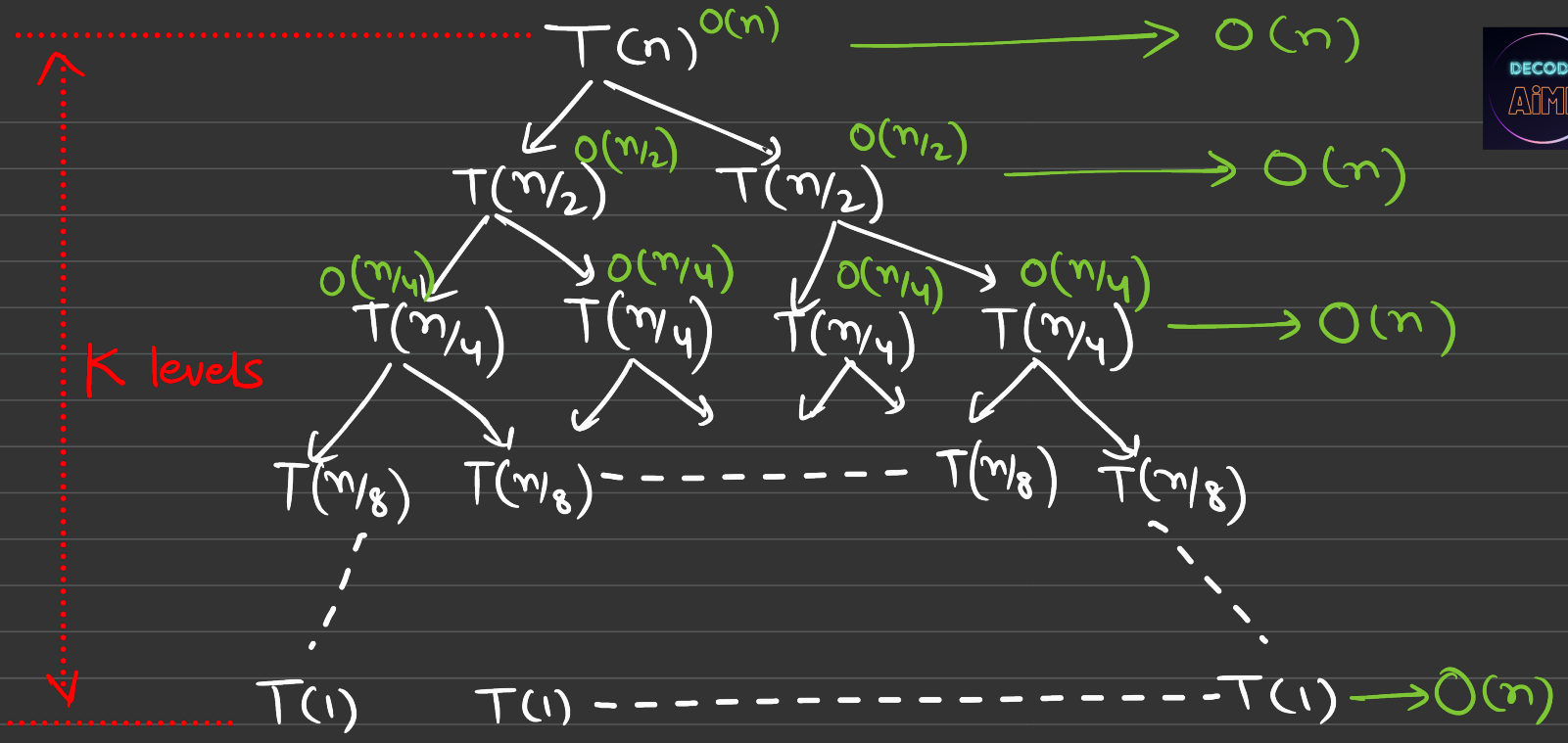
$= \theta(n \cdot \log n)$

$\boxed{T(n) = O(n \cdot \log n)}$

→ Recursion Tree method.

$T(n) = 2T(n/2) + O(n)$

$= O(1), n <= 1$



→ Calculating no of levels. (K)

$$\frac{n}{2^0}, \frac{n}{2^1}, \frac{n}{2^2}, \frac{n}{2^3} \cdots \frac{n}{2^{k-1}}$$

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

$$n = 2^{k-1}$$

$$\boxed{k = \log n + 1}$$

Total time = no of levels × Time at each level

$= \log n \times O(n)$

$$T(n) = O(n \cdot \log n)$$


DECODE
AiML
Like
Subscribe