unacademy

Algorithms and Complexity Calculation Part 3

Course on Data Structure and Algorithms Using Python

Ankush Saklecha • Lesson 13 • Dec 19, 2023

Recursive Algorithm :→

$$T(1) = 1$$

$$T(n) = \text{Calling} + \text{Cost}$$

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

```
def fun(int n):
    if n == 1:
        return 1
    Else:
        return n * fun(n-1)
```

$$T(n) = T(n-1) + \Theta(1)$$

5 * fun(4)

4 * 3 ⇒ 2
3 * 4 ⇒ 3
4 * 10 ⇒ ✗

⇒ 1

5 * 4

7 * f(6)

7 *

```
def fun(int n):
    if n==1:
        return n
    Else:
        return n/2 + f(n-1)
```

$$T(1) = 1$$

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

```
def fun(int n):
    if n==1:
        return n
    Else:
        return n + f(n/2)
```

$$T(1) = 1$$

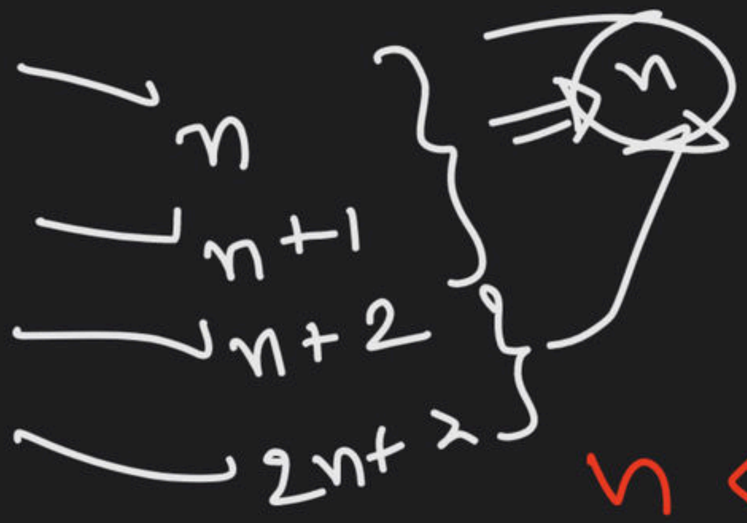
$$T(n) = T(n/2) + 1$$

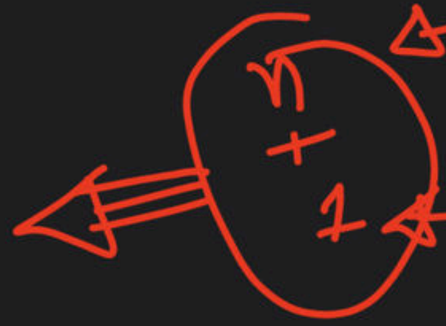
$T(1) = 1$

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

$T(n) = T(n-1) + \Theta(n)$

```
def fun(int n):
    if n==1:
        return n
    Else:
        for i in range(0, n+1):
            Print i
        return n * f(n-1)
```





$n, n+1, \frac{n}{2}, 2*n$

P1

```
def fun(int n):
    if n == 1:
        return n
    Else
        return fun(n-1) + fun(n-1)
```

$T(1) = 1$

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

or

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

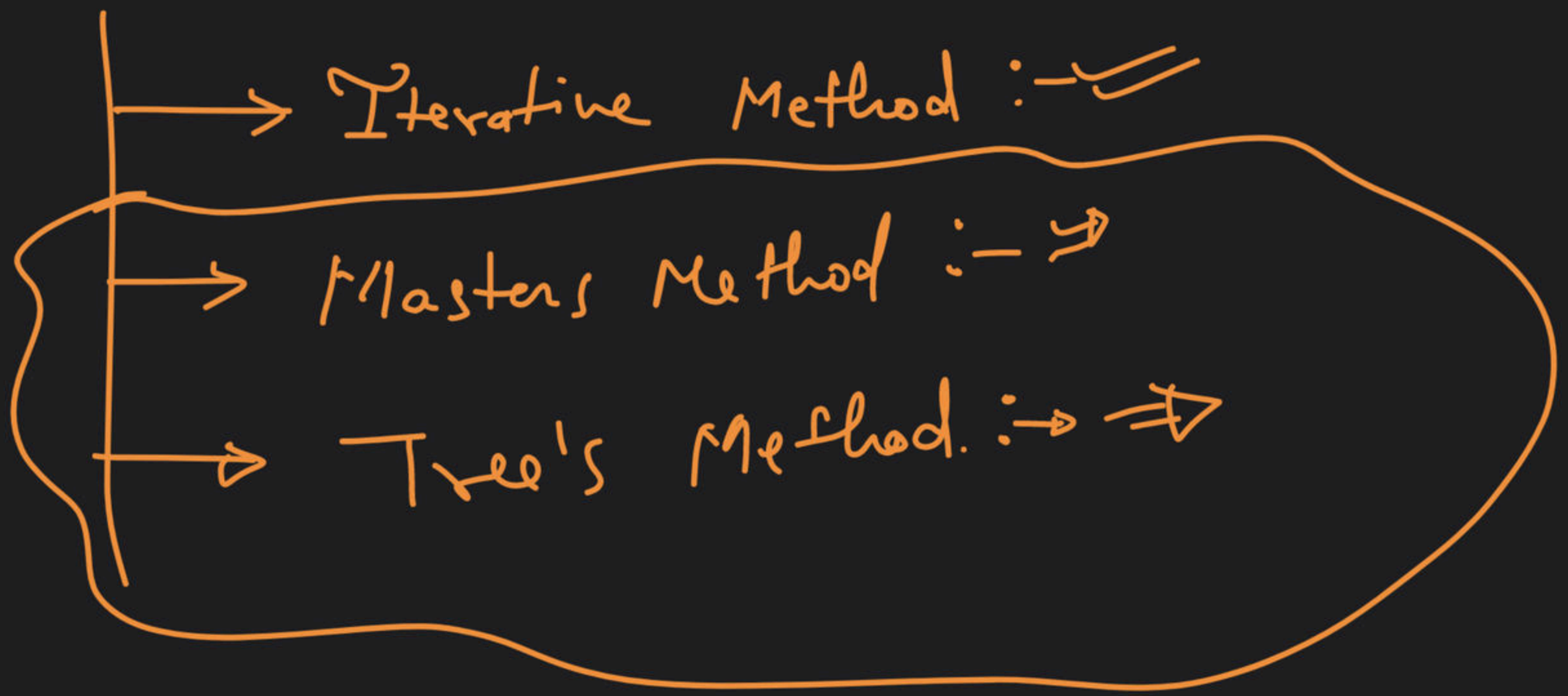
P2

```
def fun(int n):
    if n == 1:
        return n
    Else
        return 2*fun(n-1)
```

$T(1) = 1$

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

How to solve Recurrence Relations !



unacademy

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

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

$$= T(n-3) + 1 + 1 + 1$$

$$\vdots$$

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

$n - K = 1$

$K = n - 1$

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

$$= 1 + n - 1$$

$$\Rightarrow n \qquad \Theta(n)$$

$T(1) = 1$

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

unacademy

$$T(n) = T(n-1) + n \qquad T(1) = 1$$

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

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

$$= T(n-3) + (n-2) + (n-1) + n$$

$$= T(n-K) + (n-(K-1)) + \cdots\cdots (n-1) + n$$

$n-K=1$

$K=n-1$

$$= T(1) + (n-(n-1-1)) + \cdots (n-1) + n$$

$$= 1 + 2 + \cdots (n-1) + n$$

$$= \frac{n(n+1)}{2} \Rightarrow \Theta(n^2)$$

unacademy

$$T(n) = 2T(n-1) + 1 \qquad T(1) = 1$$

$$= 2\left[2T(n-2) + 1\right] + 1 \qquad T(n-1) = 2T(n-2)+1$$

$$= 2^2 T(n-2) + 2 + 1 \qquad T(n-2) = 2T(n-3)+1$$

$$= 2^2\left[2T(n-3) + 1\right] + 2 + 1$$

$$= 2^3 T(n-3) + 2^2 + 2 + 1$$

$$\vdots$$

$$2^K T(n-K) + 2^{K-1} + \cdots + 2^2 + 2 + 1$$

$n - K = 1$

$K = n - 1$

$$2^{n-1} T(1) + 2^{n-2} + \cdots + 2^2 + 2 + 1$$

Time and Space Complexity of an Algorithm

unacademy

- **Time Complexity:** The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
- The **space complexity** of an algorithm quantifies the amount of space taken by an algorithm to run as a function of the length of the input.
  - ***Auxiliary Space***
  - ***Input Space***

unacademy

Types of Algorithm

- Sequential Algorithm
- Iterative Algorithm
- Recursive Algorithm

unacademy

Master Theorem For Subtract Recurrences

- Let T(n) be a function defined on positive n as shown below:

$$T(n) \leq \begin{cases} c, & \text{if } n \leq 1, \\ aT(n-b) + f(n), & n > 1, \end{cases}$$

for some constants c, a>0, b>0 k>=0 and function f(n). If f(n) is $O(n^k)$, then

1. If a<1 then $T(n) = O(n^k)$
2. If a=1 then $T(n) = O(n^{k+1})$
3. if a>1 then $T(n) = O(n^k a^{n/b})$

$T(n) = aT(n-b) + f(n)$

a>0 b>0 n^k

$1 \Rightarrow n^0$

(a)

$$T(n) = \underset{a}{1}\,T\left(n-\underset{b}{1}\right) + \underset{f(n)}{1}$$

$a = 1 \qquad b = 1 \qquad f(n) = n^0$

Case II

$a = 1 \Rightarrow \theta(n^{0+1}) \Rightarrow \underline{\theta(n)}$

$$T(n) = 2T(n-1) + 1 \Rightarrow n^0$$

$a > 1 \Rightarrow \theta(n^k a^{n/b}) \Rightarrow \theta(1.2^{n/1})$

$\Rightarrow \underline{\theta(2^n)}$

unacademy

(1) $T(n) = T(n-2) + 1$ → $a = 1$ then $\Theta(n^{k+1}) \Rightarrow \Theta(n^{0+1}) \Rightarrow \Theta(n)$

(2) $T(n) = 0.5T(n-1) + n$ → $a < 1$ then $\Theta(n)$

(3) $T(n) = 2T(n-2) + 1$ → $a > 1$ then $\Theta(1.\sqrt{2}^{n/2}) \Rightarrow \Theta((\sqrt{2})^n)$

→ $\Theta(n^k a^{n/b})$

(4) $T(n) = 2T(n-1) + n \Rightarrow a > 1$ then $(n \cdot 2^n)$

(5) $T(n) = 2T(n-2) + n \Rightarrow a > 1 \Rightarrow$ then $(n \cdot 2^{n/2})$

(6) $T(n) = 3T(n-1) + n^2 \Rightarrow a > 1 \Rightarrow$ then $(n^2 \cdot 3^n)$

(7) $T(n) = 4T(n-2) + n^2 \Rightarrow a > 1 \Rightarrow$ then $(n^2 \cdot 4^{n/2})$

$n^2 \cdot 4^{n/2}$

unacademy

Master's Method for Divide and Conquer

$$T(n) = aT(n/b) + f(n) \cdot (\log n)^K$$

$a > 0$, $b > 1$, $K \geq 0$, $f(n)$ = Any Polynomial function

↳ all are +ve constant

n^i

(I) $n^{\log_b a} > f(n) \Rightarrow \Theta(n^{\log_b a})$

(II) $n^{\log_b a} < f(n) \Rightarrow \Theta(f(n) \cdot (\log n)^K)$

(III) $n^{\log_b a} = f(n) \Rightarrow \Theta(f(n) \cdot (\log n)^{K+1})$

unacademy

$$T(n) = T(n/2) + n \quad \text{---}$$

$a = 1 \quad b = 2 \quad f(n) = n \quad k = 0$

$$n^{\log_2^1} \Rightarrow n^0 \Rightarrow 1 < n$$

$$\Theta(n)$$

unacademy

$$T(n) = 2T(n/2) + 1$$

$a = 2 \quad b = 2 \quad f(n) = 1 \quad , K = 0$

$$n^{\log_2 2} \Rightarrow n^1 > 1$$

$\log_b 9 \Rightarrow \frac{\log_{10} 9}{\log_{10} b}$

$\rightarrow \log_5 9$

$\Theta(n)$

unacademy

(1) $T(n) = T(n/2) + 1$ → $n^{\log_2 1} \Rightarrow n^0 \Rightarrow 1 \equiv 1 \Rightarrow \Theta(1.(\log n)^{0+1})$

$\Rightarrow \Theta(\log n)$

(2) $T(n) = 2T(n/2) + 1$ → $\Theta(n)$

(3) $T(n) = 4T(n/2) + n$ → $n^{\log_2 4} \Rightarrow n^2 > n \Rightarrow \Theta(n^2)$

(4) $T(n) = T(n/2) + 1.\log n$ → $n^{\log_2 1} \Rightarrow n^0 \Rightarrow 1 \equiv 1 \Rightarrow \Theta(1.(\log n)^{1+1})$

$\Rightarrow \Theta((\log n)^2)$

(5) $T(n) = 2T(n/2) + n\log n$

↳ $n^{\log_2 2} \Rightarrow n \equiv n \Rightarrow \Theta(n(\log n)^2)$

unacademy

(1) $T(n) = 2T(n/4) + \sqrt{n}\log n$ → $n^{\log_4 2} \Rightarrow n^{\frac{1}{\log_2 4}} \Rightarrow n^{\frac{1}{2}} \Rightarrow \sqrt{n} \equiv \sqrt{n}$

(2) $T(n) = 8T(n/2) + n^2$ → $n^{\log_2 8} \Rightarrow n^3 > n^2 \Rightarrow \Theta(n^3)$

$\Theta(\sqrt{n}(\log n)^2)$

(3) $T(n) = 2T(n/8) + n$ → $n^{\log_8 2} \Rightarrow n^{\frac{1}{\log_2 8}} \Rightarrow n^{1/3} < n \Rightarrow \Theta(n)$

(4) $T(n) = 1T(n/2) + n^2$ → $n^{\log_2 1} \Rightarrow n^0 < n^2 \Rightarrow \Theta(n^2)$

(5) $T(n) = 4T(n/2) + n^2$ → $n^{\log_2 4} \Rightarrow n^2 \equiv n^2$

→ $\Theta(n^2 \log n)$

unacademy

Extended Master's Method 1

$T(n) = aT(n/b) + \frac{f(n)}{\log n}$

- $T(n) = aT(n/b) + f(n)\,(\log n)^k$
- Here $f(n)$ is polynomial function
- a, b are positive constant $a > 0, b > 1, k = -1$

- Calculate $n^{\log_b a}$

Case 1: If $f(n) < n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(n^{\log_b a})$

Case 2: If $f(n) > n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(f(n))$

Case 2: If $f(n) = n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(f(n)\log\log n)$

unacademy

Extended Master's Method 2

- $T(n) = aT(n/b) + f(n)\,(\log n)^k$
- Here $f(n)$ is polynomial function
- a, b, k are positive constant $a > 0, b > 1, k < -1$

- Calculate $n^{\log_b a}$

Case 1: If $f(n) < n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(n^{\log_b a})$

Case 2: If $f(n) > n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(f(n))$

Case 3: If $f(n) = n^{\log_b a}$ then $T(n) = \boldsymbol{\theta}(n^{\log_b a})$

(1) $T(n) = 1 \cdot T(n/2) + 1 \cdot (\log n)^{-1}$ → $n^{\log_2 1} \Rightarrow n^0 \Rightarrow 1$ $\Theta(1 \cdot \log\log n)$

(2) $T(n) = 2T(n/2) + 1 \cdot (\log n)^{-2}$ → $n^{\log_2 2} \Rightarrow n > 1 \Rightarrow \Theta(n)$

(3) $T(n) = 4T(n/2) + \frac{n}{\log n}$ → $n^{\log_2 4} \Rightarrow n^2 > n \Rightarrow n^2$

(4) $T(n) = T(n/2) + \frac{1}{(\log n)^2}$ → $n^{\log_2 1} \Rightarrow n^0 \Rightarrow 1 = 1 \Rightarrow \Theta(1)$

(5) $T(n) = 2T(n/2) + \frac{n}{\log n}$ → $n^{\log_2 2} \Rightarrow n = n \Rightarrow \Theta(n \log\log n)$

(6) $T(n) = 2T(n/4) + \frac{\sqrt{n}}{(\log n)^3}$ → $n^{\log_4 2} \Rightarrow \sqrt{n} = \sqrt{n} \Rightarrow \Theta(\sqrt{n})$

(7) $T(n) = 8T(n/2) + \frac{n^2}{\log n}$ → $n^{\log_2 8} \Rightarrow n^3 > n^2 \Rightarrow \Theta(n^3)$

unacademy

- Recursive Relation
  - Substitution Method
  - Trees Method

Trees Method :→

$$T(n) = aT(n/b) + f(n)(\log n)^k$$

↳→ Here we can apply Master's.

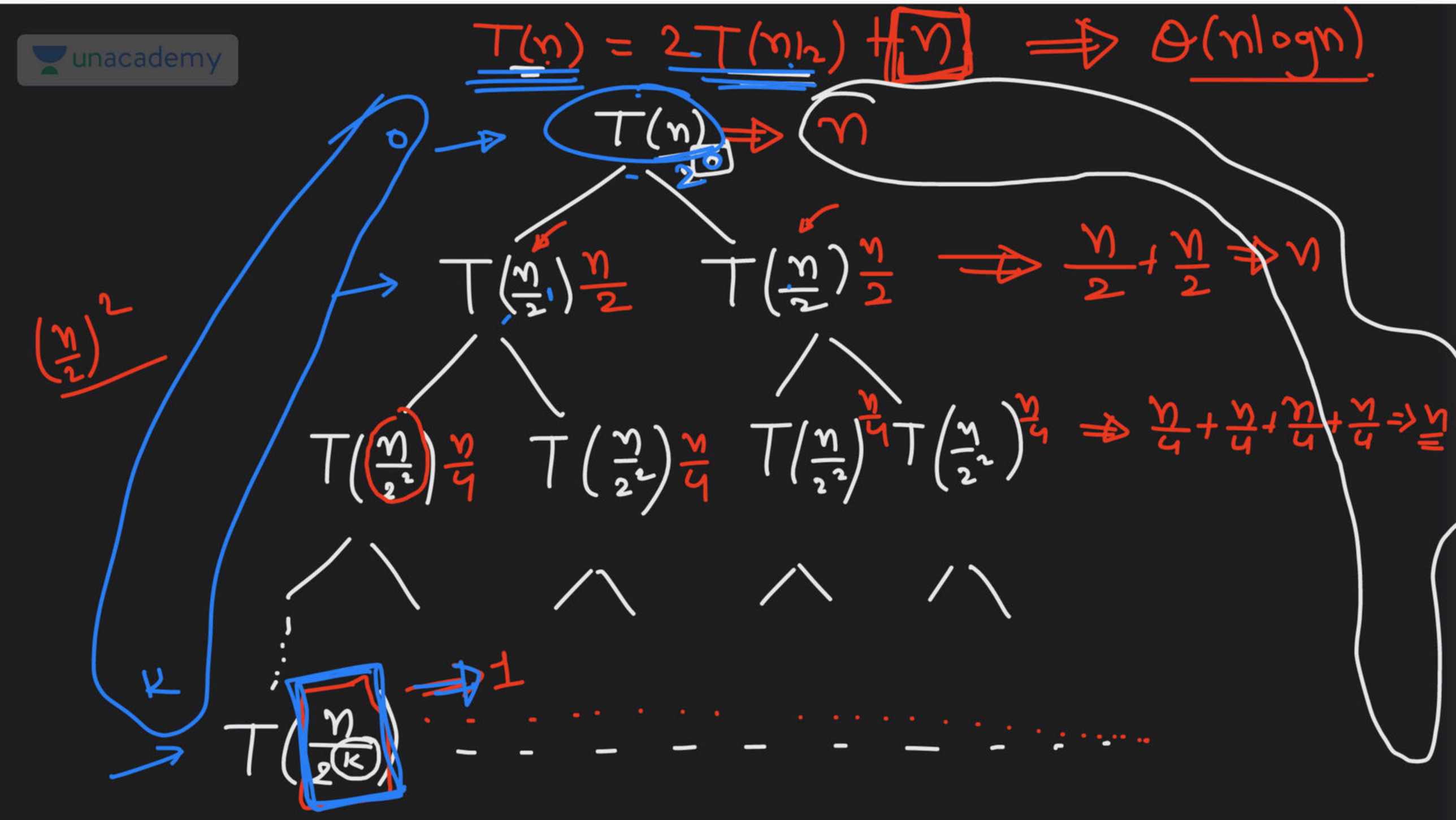
$$T(n) = T(n/4) + T(3n/4) + n$$

↳ we can't apply Masters.

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

$$\Downarrow$$

$$\boxed{T(n) = 2T(n/2) + n}$$


T(n) = 2T(n/2) + n ⇒ Θ(nlogn)
T(n)
n
0
2^0
T(n/2) n/2
T(n/2) n/2
n/2 + n/2 ⇒ n
(n/2)^2
T(n/2^2) n/4
T(n/2^2) n/4
T(n/2^2) n/4
T(n/2^2) n/4
n/4 + n/4 + n/4 + n/4 ⇒ n
k
T(n/2^k)
1

unacademy

$n + n + n + \ldots\ldots\ldots (k+1)$ times

$(k+1)\, n$

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

$n = 2^k$

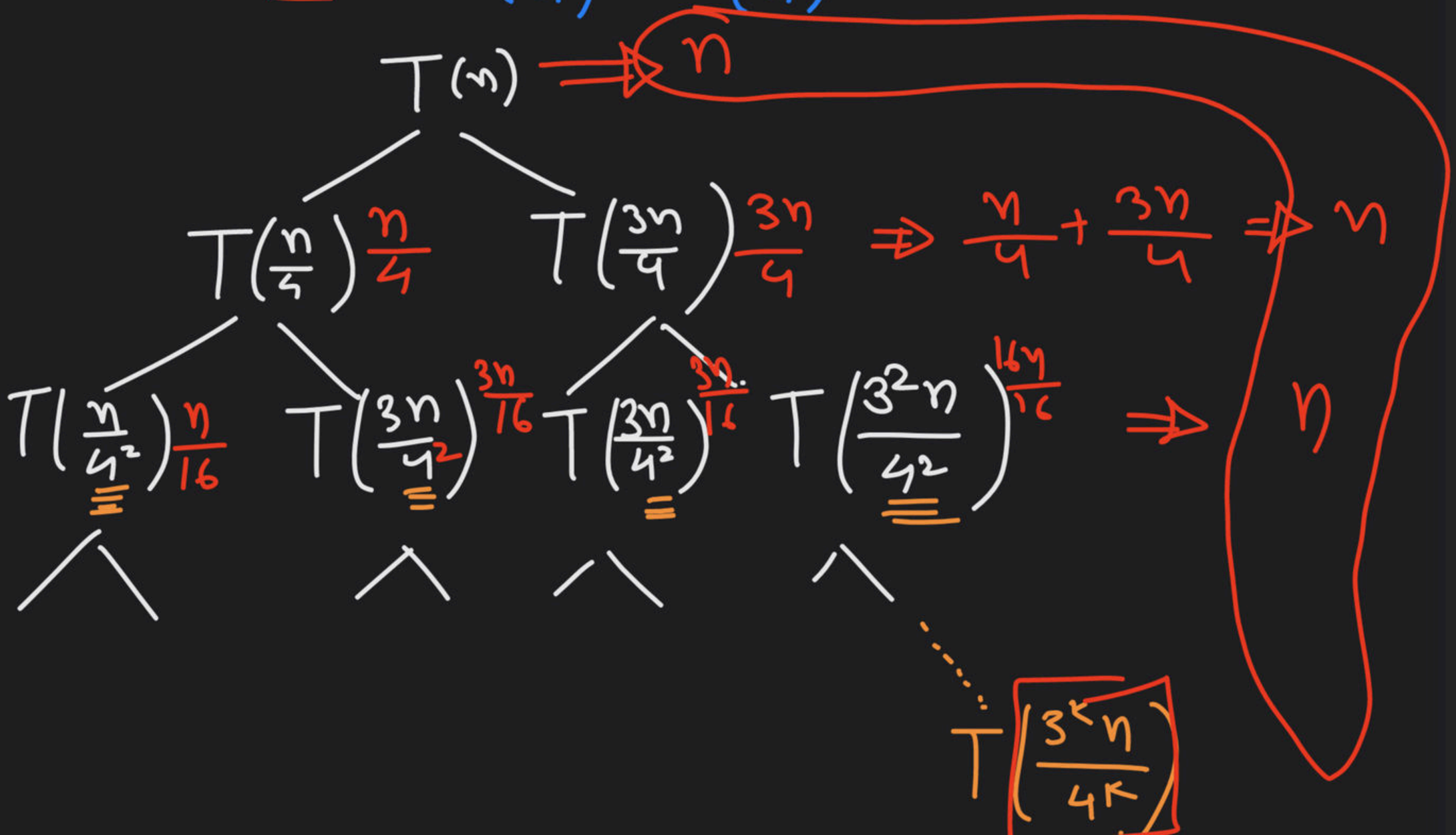
$k = \log_2 n$

$(\log n + 1)\, n$

$n \log n + n$

$\Theta(n \log n)$

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n$$



unacademy

$n + n + \cdots (k+1)$ times

$n(k+1)$

$n\left(\log_{4/3} n + 1\right)$

$n \log_{4/3} n + n$

$O(n \log n)$

$\frac{3^k n}{4^k} = 1$

$n = \left(\frac{4}{3}\right)^k$

$\log n = \log\left(\frac{4}{3}\right)^k$

$\log n = k \log_{4/3}$

$k = \frac{\log n}{\log_{4/3}}$

$k = \log_{4/3} n$

unacademy

$$T(n) = 2T(\lfloor n \rfloor / 2) + \sqrt{n}$$

$$T(n) = 2T(n/2) + \sqrt{n}$$

$$n^{\log_2 2} \Rightarrow n > \sqrt{n}$$

$$\Theta(n)$$

Base 2 ← $T(n) = 2T(n/2) + n$ ✓

$3/2$ ← $T(n) = T(n/3) + T(\frac{2n}{3}) + n$ ✓

$4/3$ ← $T(n) = T(n/4) + T(\frac{3n}{4}) + n$ ✓

$5/4$ ← $T(n) = T(n/5) + T(\frac{4n}{5}) + n$ ✓

$\Rightarrow \Theta(n \log n)$

unacademy

$$T(n) = T(n/3) + T(n/4) + n^2$$

$$T(n) \Rightarrow n^2$$

$T(\frac{n}{3}) \Rightarrow (\frac{n}{3})^2$ $\quad T(\frac{n}{4}) \Rightarrow (\frac{n}{4})^2 \Rightarrow \frac{n^2}{9} + \frac{n^2}{16} \Rightarrow \frac{25n^2}{144}$

$T(\frac{n}{9})$ $(\frac{n}{9})^2$ $\quad T(\frac{n}{12})$ $(\frac{n}{12})^2$ $\quad T(\frac{n}{12})$ $(\frac{n}{12})^2$ $\quad T(\frac{n}{4^2})$ $(\frac{n}{16})^2$ $\Rightarrow$

$$\frac{n^2}{81} + \frac{n^2}{144} + \frac{n^2}{144} + \frac{n^2}{256} \Rightarrow \left(\frac{25}{144}\right)^2 n^2 \Longrightarrow$$

$$n^2 + \frac{25n^2}{144} + \left(\frac{25}{144}\right)^2 n^2 + \cdots\cdots$$

$$n^2 \cdot \left[1 + \frac{25}{144} + \left(\frac{25}{144}\right)^2 + \cdots\cdots \left(\frac{25}{144}\right)^K + \cdots\right]$$

ignore

G.P.

$r \Rightarrow \frac{25}{144} \Rightarrow < 1$

$\Theta(n^2)$

unacademy

$$T(n) = T(\sqrt{n}) + 1$$

$$T(n) = T(n^{1/2}) + 1$$

$$= T(n^{1/4}) + 1 + 1$$

$$= T(n^{1/8}) + 1 + 1 + 1$$

$$= T(n^{\frac{1}{2^3}}) + 3$$

$$\vdots$$

unacademy

$$= T\left(n^{\frac{1}{2^k}}\right) + k$$

$$= T(c) + k$$

$$= 1 + k$$

$$1 + \log\log n$$

$$\Rightarrow O(\log\log n)$$

$$T(c) = 1$$

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

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

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

$$\left\{\frac{1}{2^k} = \frac{\log c}{\log n}\right\}$$

$$2^k = \frac{\log n}{\log c}$$

$$\Rightarrow 2^k = \log_c n$$

$$k = \log_2 \log_c n$$

$$\Rightarrow \log\log n$$

unacademy

$k = \log n$

$n = 2^k$

$T(n) = T(\sqrt{n}) + 1$

$T(2^k) = T(2^{k/2}) + 1$

$T(2^k) = S(k)$

$S(k) = S(k/2) + 1$

$k^{\log_2 1} \Rightarrow k^0 \Rightarrow 1 \equiv 1 \Rightarrow \underline{\Theta(\log k)}$

$T(2^k) = \log k$

$T(n) \Rightarrow \log \log n$

$T(n) = 2T(\sqrt{n}) + 1$

$n = 2^k$

$T(2^k) = 2T(2^{k/2}) + 1$

$T(2^k) = S(k)$

$S(k) = 2S(k/2) + 1$

$k = \log n$

$k^{\log_2 2} \Rightarrow k^1 \Rightarrow 1 \Rightarrow O(k)$

$\Rightarrow O(\log n)$

$T(2^k) = k$

$T(n) = \log n$

$$n = 2^k$$

$$\log n = \log 2^k$$

$$\log n = k \log 2$$

$$\frac{\log n}{\log 2} = k \Rightarrow \log_2 n = k$$

How to Calculate Space Complexity of an Algorithm

unacademy

Space complexity *is the amount of memory used by the algorithm to execute and produce the result.*

- ***Auxiliary Space***
- ***Input Space***

unacademy

Space Complexity

- How to calculate Space complexity of
  - Sequential Algorithm
  - Iterative Algorithm
  - Recursive Algorithm

Algorithms

- Sequential Algorithm

Algorithms

- Iterative Algorithm

```
def initialize_array(A, n):
    for i in range(n):
        A[i] = 0

# Example usage:
n = 10  # Replace with the desired array size
A = [0] * n  # Initialize an array of size n with zeros
initialize_array(A, n)
print(A)  # You can print the array to verify the initialization
```

```python
# Example array A
A = [1, 2, 3, 4, 5]  # Replace with your array

n = len(A)
B = [0] * n  # Initialize array B with zeros
C = [0] * n  # Initialize array C with zeros

for i in range(n):
    B[i] = A[i]
    C[i] = A[i]

# Printing the results
print("Array B:", B)
print("Array C:", C)
```

```python
# Example array A
A = [1, 2, 3, 4, 5]  # Replace with your array
n = len(A)

# Create an empty 2D array C with dimensions n x n
C = [[0 for _ in range(n)] for _ in range(n)]

# Copy the elements from A to C
for i in range(n):
    for j in range(n):
        C[i][j] = A[i]

# Printing the resulting 2D array C
for row in C:
    print(row)
```

Algorithms

- Recursive Algorithm

unacademy

```
def f(n):
    if n / 2 <= 1:
        return n
    f(n / 2)
    f(n / 2)

# Example usage
result = f(8)  # Replace 8 with the desired value
print(result)
```

Algo 1:

```python
def fun(n):
    if n <= 1:
        return n
    else:
        return fun(n - 1) + fun(n - 1)

# Example usage:
result = fun(5)  # Replace 5 with the desired value
print(result)
```

Algo 2:

```python
def fun(n):
    if n <= 1:
        return n
    else:
        return 2 * fun(n - 1)

# Example usage:
result = fun(5)  # Replace 5 with the desired value
print(result)
```

unacademy

```
int fun( int n)
{
if(n<=1)
return n;
else
{
fun(n-1);
for(i=1;i<=n; i=i+10);
return i;
}
}
```

```python
def fun(n):
    if n <= 1:
        return n
    else:
        fun(n - 1)
        i = 1
        while i <= n:
            i += 10
        return i

# Example usage:
result = fun(5)  # Replace 5 with the desired value
print(result)
```

unacademy

```
int fun( int n)
{
if(n<=1)
return n;
else
{
fun(n/2);
for(i=1;i<=n; i=i+10);
return i;
}
}
```

```python
def fun(n):
    if n <= 1:
        return n
    else:
        fun(n // 2)  # Note: Use integer division (//) to match C++ behavior
        i = 1
        while i <= n:
            i += 10
        return i

# Example usage:
result = fun(20)  # Replace 20 with the desired value
print(result)
```

What is Space Complexity?

(a) $\theta(n)$

(b) $\theta(\log n)$

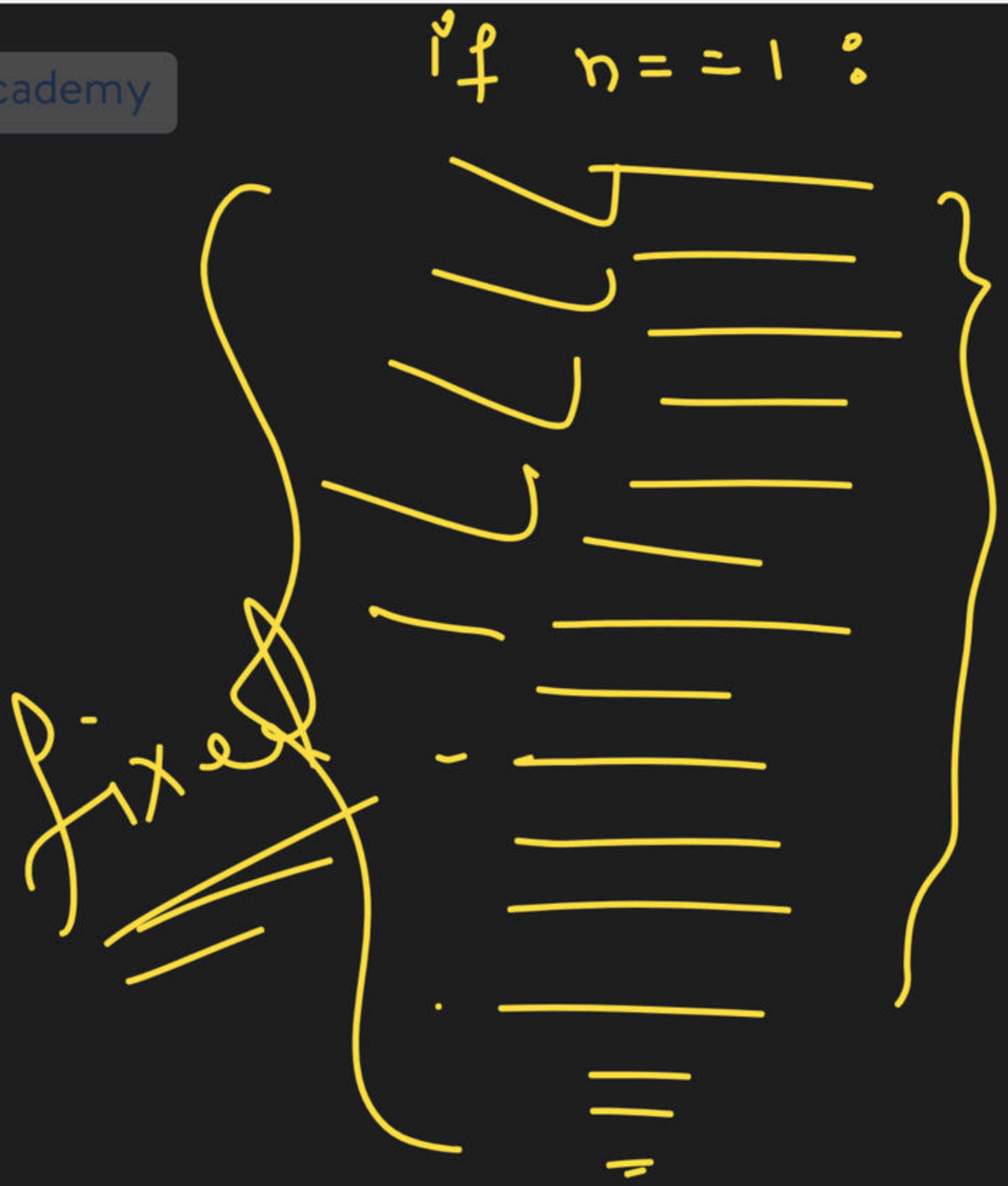
(c) $\theta(n \log n)$

(d) None

$$= 2^0 + 2 + 2^2 + \text{- - - - } 2^{n-1}$$

$$= \frac{1.(2^n - 1)}{(2-1)} \Rightarrow 2^n - 1$$

$$\Rightarrow \underline{O(2^n)}$$



$T(1) = 1$