unacademy

Sorting techniques Part 1

Course on Data Structure and Algorithms Using Python

Ankush Saklecha • Lesson 14 • Dec 20, 2023

unacademy

Time Complexity

1. Time complexity of an algorithm T(n), where n is the input size is given by $T(n) = T(n-1) + 1/n$, if $n>1$ otherwise $T(n) = 1$. The order of algorithm is

(A) Log n
(B) N
(C) N^2
(D) N^n

$$T(n) = T(n-1) + \frac{1}{n}$$

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

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

$$\vdots$$

$$T(n-k) + \frac{1}{n-(k-1)} + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

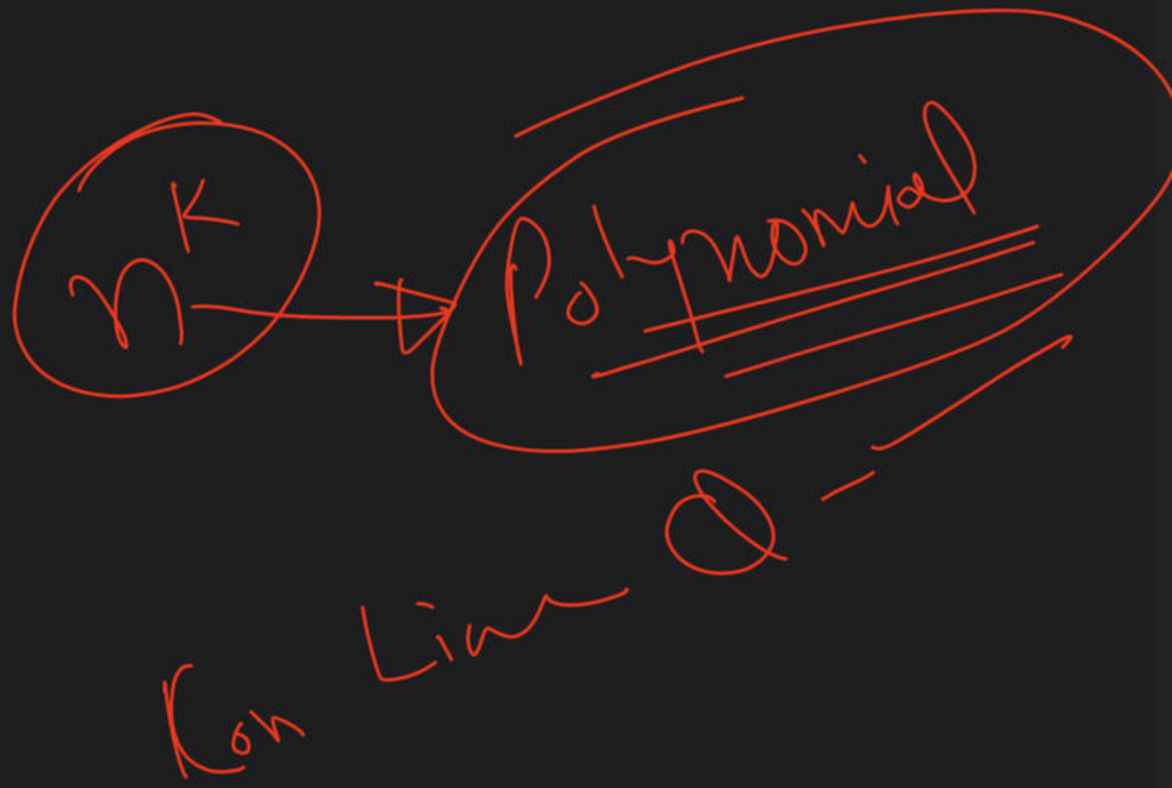
$$T(1) + \frac{1}{n-(n-1-1)} + \cdots \frac{1}{n-1} + \frac{1}{n}$$

$n-1 = 1$

$k = n-1$

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$\hookrightarrow \log n$


n^k
Polynomial
For Limit Q -

2. Solve the following recurrence relations
$T(n) = T(n-1) + n/2, T(1) = 1$

(A) $\Theta(\log n)$ (B) $\Theta(n \log n)$

(C) $\Theta(n^2)$ (D) $\Theta(n)$

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

$a = 1$ then $\Theta(n^{k+1})$

$\Theta(n^2)$

3. Given asymptotic notation in each of the following recurrences.

a. $T(n) = 2T(n/2) + n^3$

(A) $\Theta(n)$ (B) $\Theta(n^3)$ (C) $\Theta(n^3 \log n)$ (D) None

$n^{\log_2 2} \Rightarrow n^1 < n^3$

b. $T(n) = T(9n/10) + n$

(A) $\Theta(n)$ (B) $\Theta(n^{1/2})$ (C) $\Theta(n^3 \log n)$ (D) None

$n^{\log_{10/9} 1} \Rightarrow n^0 \Rightarrow 1 < n$

c. $T(n) = 16T(n/4) + n^2$
(A) $\Theta(n)$ (B) $\Theta(n\log n)$ (C) $\Theta(n^2\log n)$ (D) None

$n^{\log_4 16} \Rightarrow n^{\log_4 4^2} \Rightarrow n^{2\log_4 4} \Rightarrow n^2 \equiv n^2$

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

d. $T(n) = 3T(n/2) + n\lg n$.
(A) $\Theta(n)$ (B) $\Theta(n^{3/2})$ (C) $\Theta(n^{1/2}\log n)$ (D) $\Theta(n^{1/2})$

$n^{\log_2 3} \Rightarrow n^{1.5} > n$ $\Theta(n^{1.5})$

$T(n) = T(n-1) + n + \log n \rightarrow T(n) = T(n-1) + n$

c. $T(n) = T(n-1) + \lg n$.
(A) $\Theta(n)$ (B) $\Theta(2^n)$ (C) $\Theta(n\log n)$ (D) $\Theta(\log n)$

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

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

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

$\vdots$

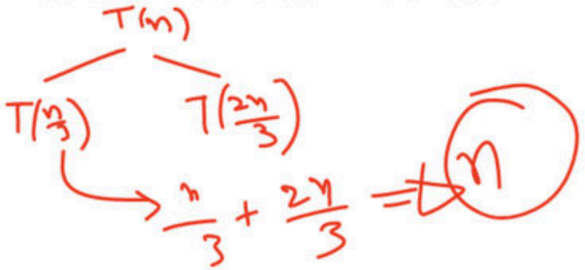
$= T(n-k) + \log(n-(k-1)) + \cdots + \log n$

$n-k=1$

$k=n-1$

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

f. $T(n) = T(n/3) + T(2n/3) + n$.
(A) $\Theta(n)$ (B) $\Theta(2^n)$ (C) $\Theta(n\log n)$ (D) $\Theta(\log n)$



$\log 1 + \log 2 + \cdots \log n$

$\log(1 \times 2 \times 3 \times \cdots n)$

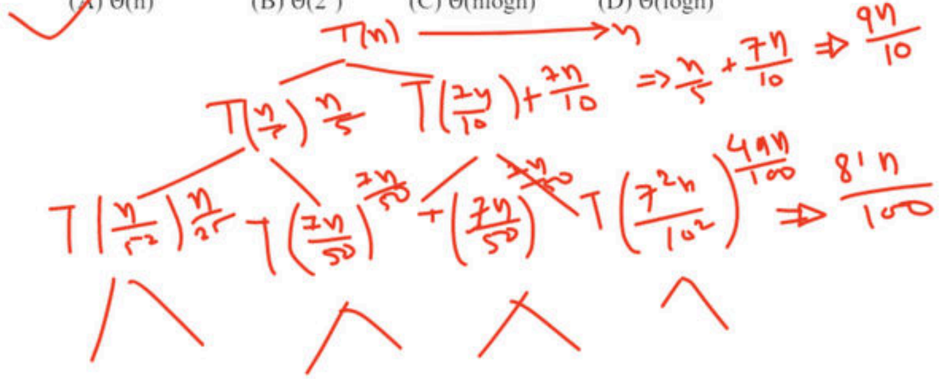
$\log(n!) \Leftrightarrow \Theta(\log(n!))$

$\log n^n$ ⇒ $n \log n$

$n! \cong n^n$

g. $T(n) = T(n/5) + T(7n/10) + n$

(A) $\Theta(n)$ (B) $\Theta(2^n)$ (C) $\Theta(n\log n)$ (D) $\Theta(\log n)$



$$n + \frac{9n}{10} + \frac{9^2 n}{10^2} + \cdots$$

$$n\left[1 + \frac{9}{10} + \frac{9^2}{10^2} + \cdots\right]$$

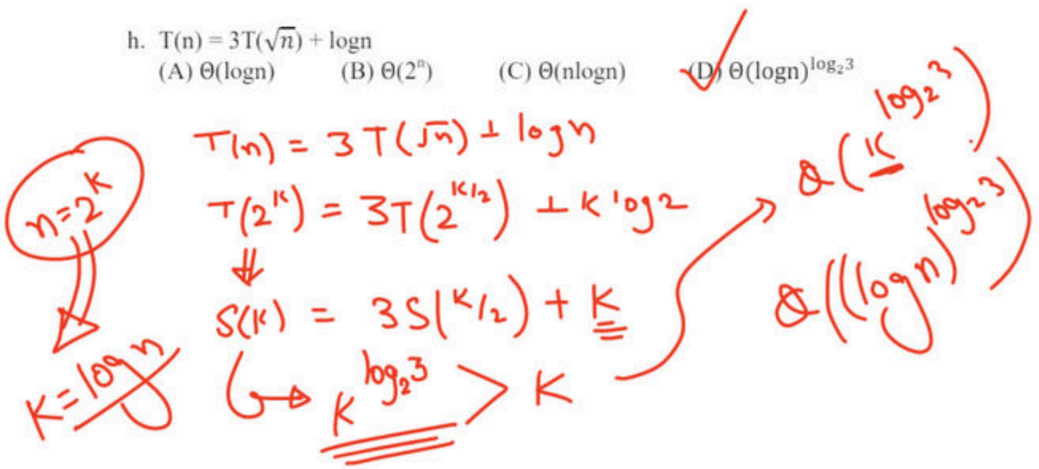
Ignore

$$r = \frac{9}{10} < 1$$

n

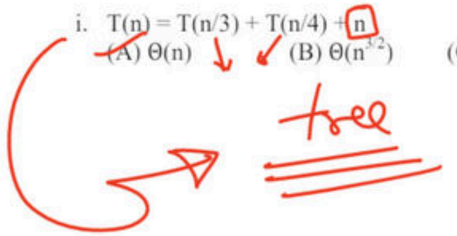
h. $T(n) = 3T(\sqrt{n}) + \log n$

(A) $\Theta(\log n)$ (B) $\Theta(2^n)$ (C) $\Theta(n \log n)$ (D) $\Theta(\log n)^{\log_2 3}$

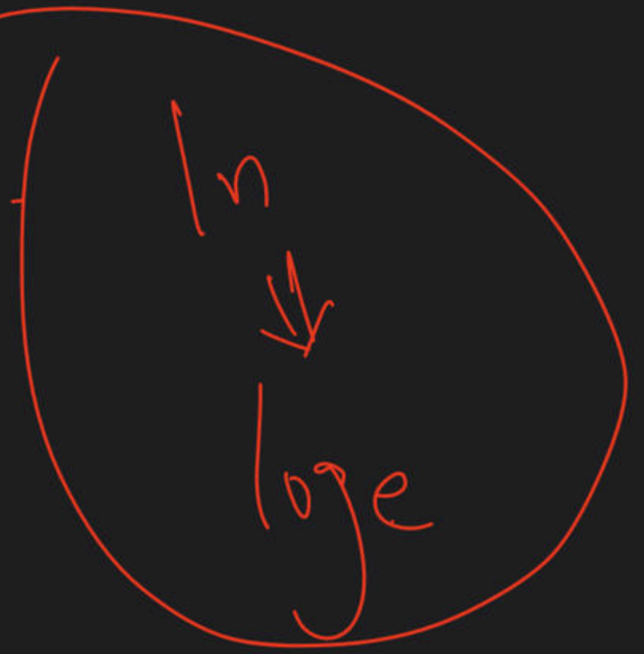


i. $T(n) = T(n/3) + T(n/4) + n$

(A) $\Theta(n)$ (B) $\Theta(n^{3/2})$ (C) $\Theta(n^{1/2} \log n)$ (D) $\Theta(n^{1/2})$




log


ln
loge

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 $T(n) = O(n^k a^{n/b})$

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

(2) $T(n) = 0.5T(n-1) + n$

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

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

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

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

(7) $T(n) = 4T(n-2) + n^2$

unacademy

Master’s Method for Divide and Conquer

unacademy

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

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

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

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

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

unacademy

(1) $T(n) = 2T(n/4) + \sqrt{n}logn$

(2) $T(n) = 8T(n/2) + n^2$

(3) $T(n) = 2T(n/8) + n$

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

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

Extended Master's Method 1

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

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

Extended Master's Method 2

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

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

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

(2) $T(n) = 2T(n/2) + (\log n)^{-2}$

(3) $T(n) = 4T(n/2) + \frac{n}{\log n}$

(4) $T(n) = T(n/2) + \frac{1}{(\log n)^2}$

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

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

(7) $T(n) = 8T(n/2) + \frac{n^2}{\log n}$

unacademy

- Recursive Relation
  - Substitution Method
  - Trees Method

$$a^{\log_a b} \Rightarrow b$$

$$\log_b a = \frac{\log a}{\log b} = \frac{1}{\log_a b}$$

$$\log m^n = n \log m$$

$$\log(m \times n) = \log m + \log n$$

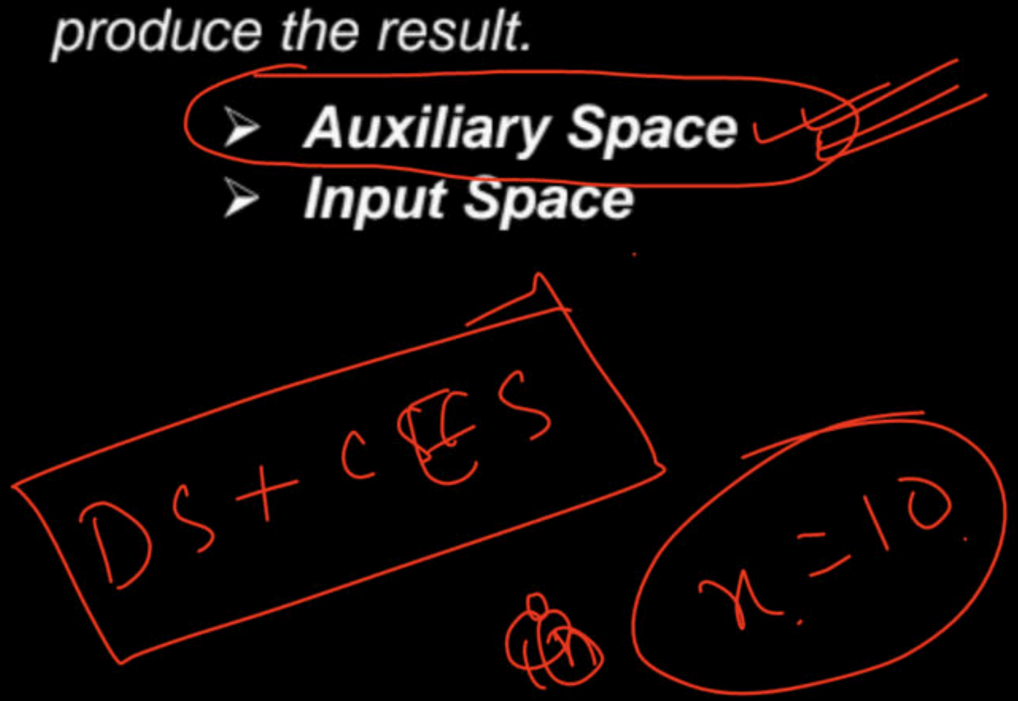
$$\log\left(\frac{m}{n}\right) = \log m - \log n$$

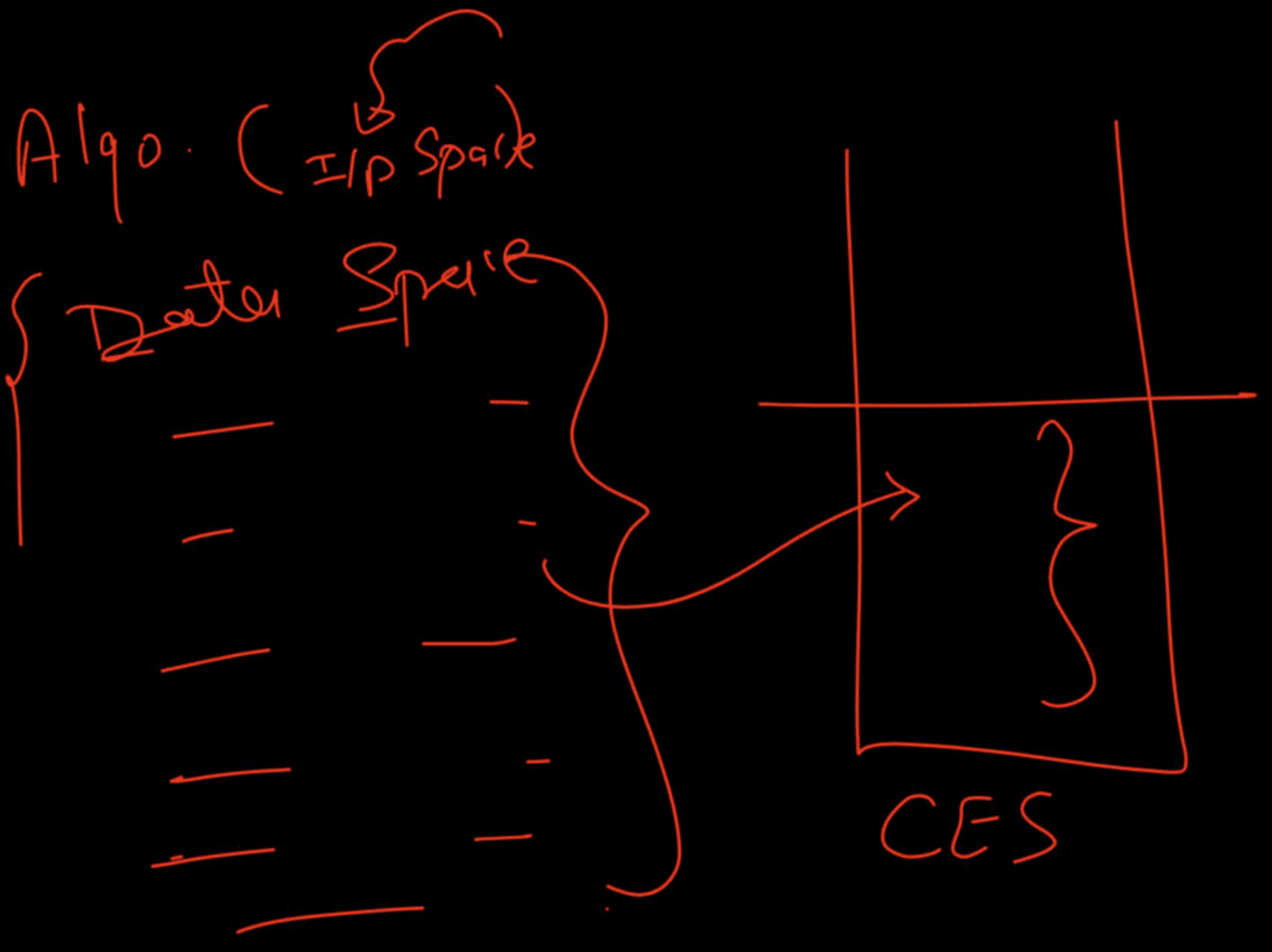
How to Calculate Space Complexity of an Algorithm

unacademy

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

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



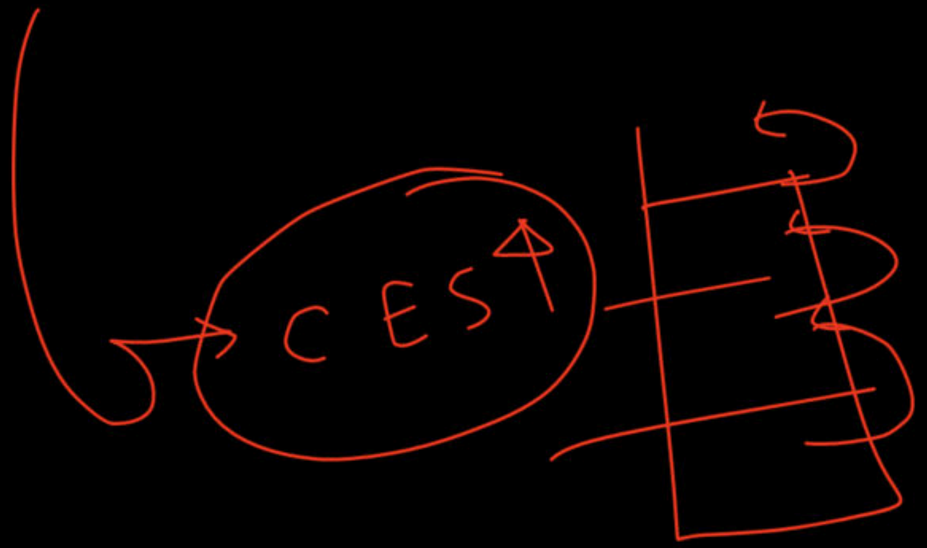


unacademy

Space Complexity

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

Code execution Spaie ⇒ 1 (Fixed)



Algorithms

- Sequential Algorithm

Algorithms

- Iterative Algorithm

```python
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
```

CSE + DS

1 + 1 → 1

Space ⇒ O(1)

Time = O(n)

Input Space = Array of size n

```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)
```

Time | $O(n)$

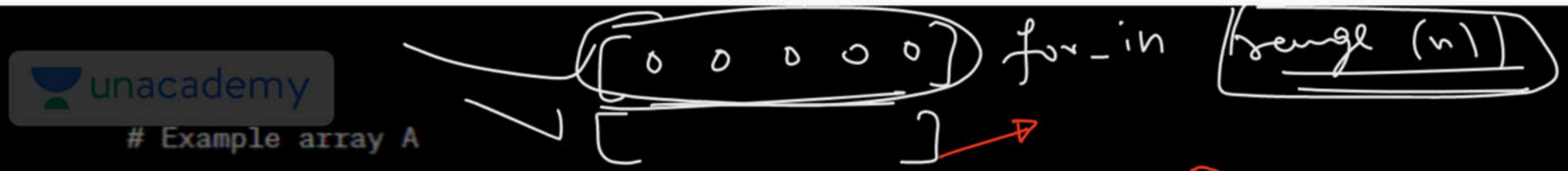
Space $\Rightarrow$ CES + DS

$1 + n + n + n$

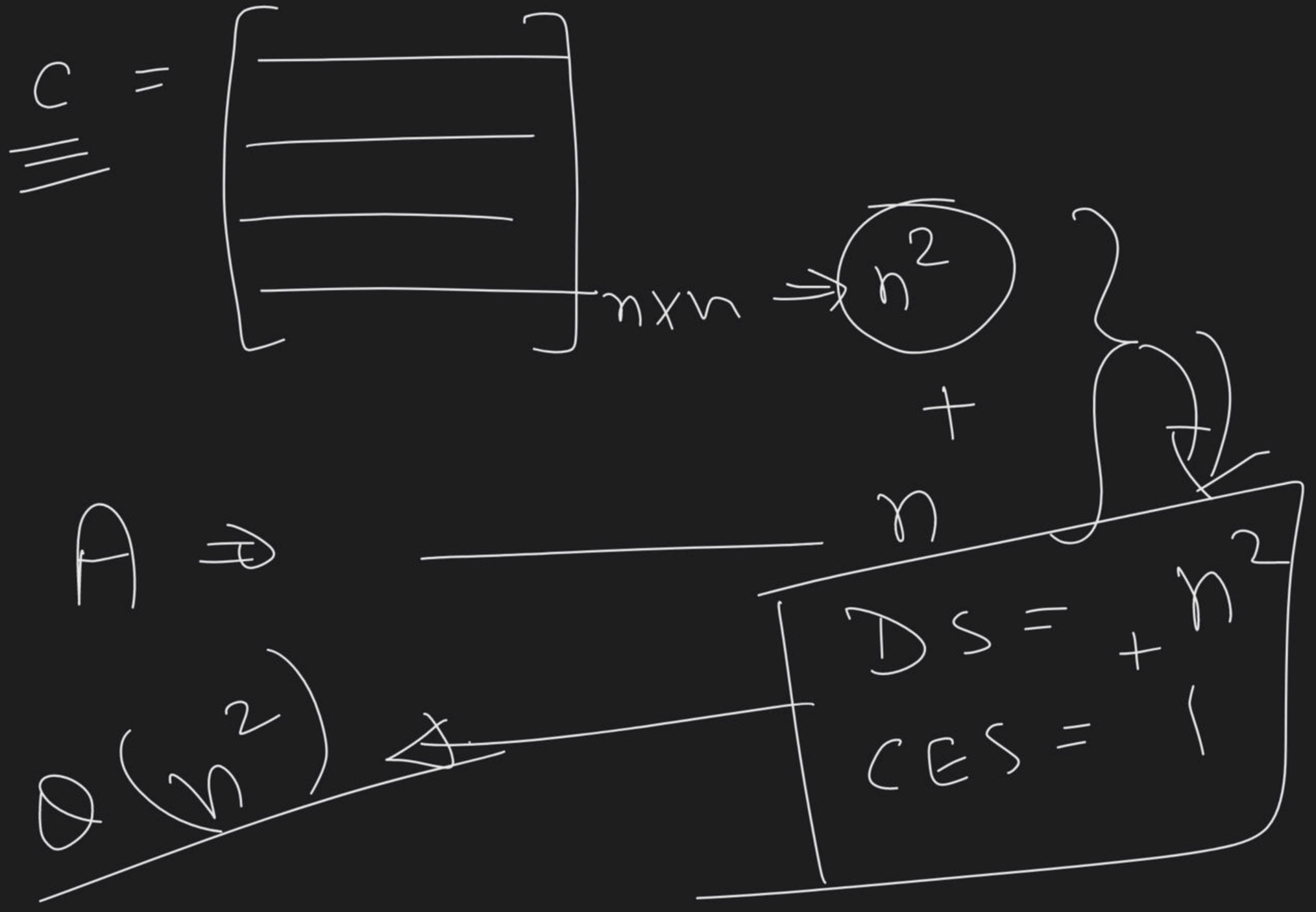
$\Rightarrow 1 + 3n$

$\Rightarrow n$

$O(n)$




```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)
```

2

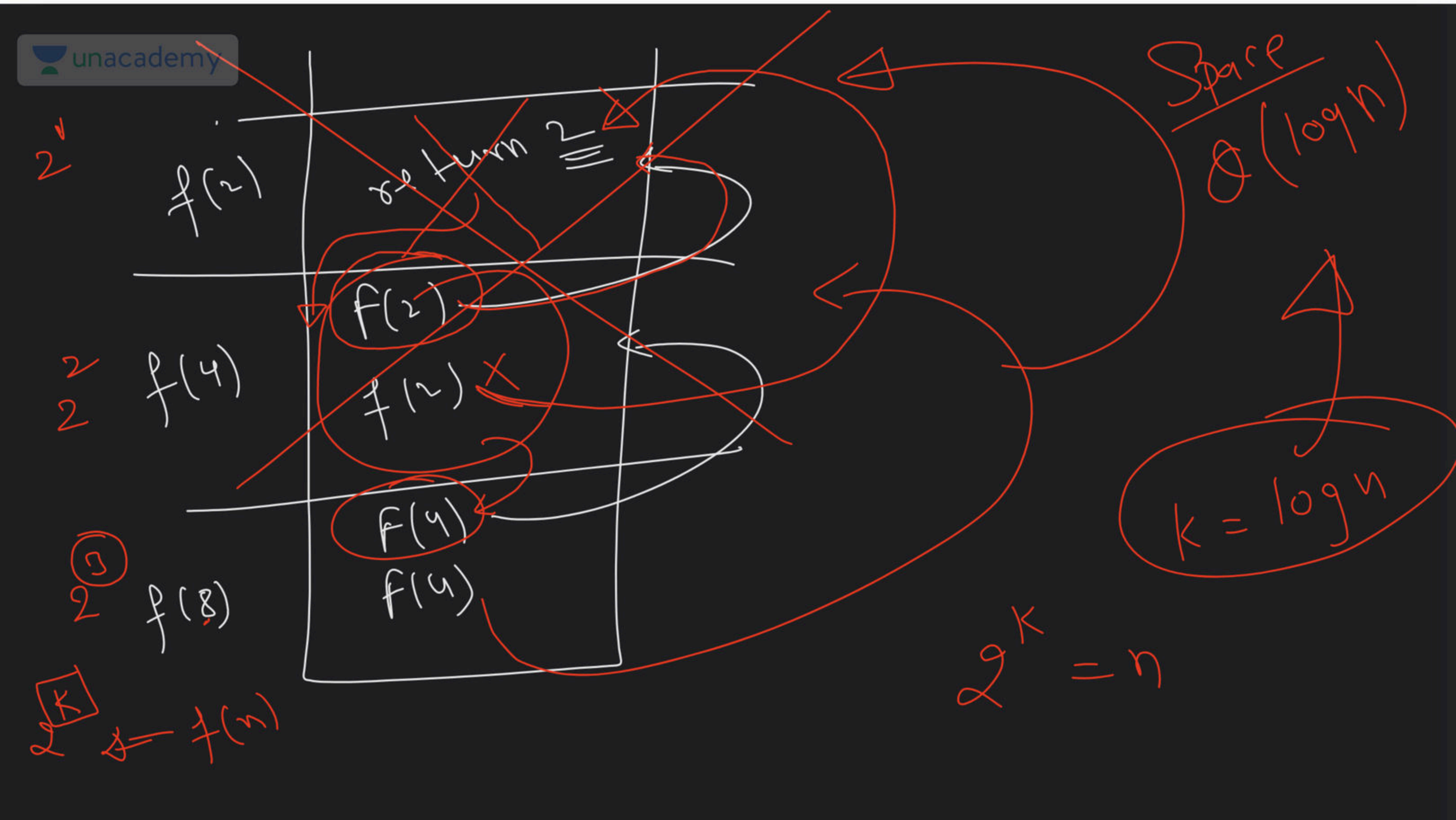
Time $\Rightarrow T(n) = 2T(n/2) + 1$

$\hookrightarrow$ @ $n^{\log_2 2} > 1$

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

$T(n) = \underline{\underline{\text{calling}}} + \underline{\underline{\text{cost}}}$

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


unacademy
f(2)
return 2
f(2)
f(1)
f(4)
f(1)
f(4)
f(4)
f(8)
Space
O(log n)
k = log n
2^k = n

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)
```

1

[1] o/p (for n=5)

[2] Time (For n)

[3] Space. (for n) $T(1) = 1$

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

$\hookrightarrow \Theta(2^n)$.

Algo 2:

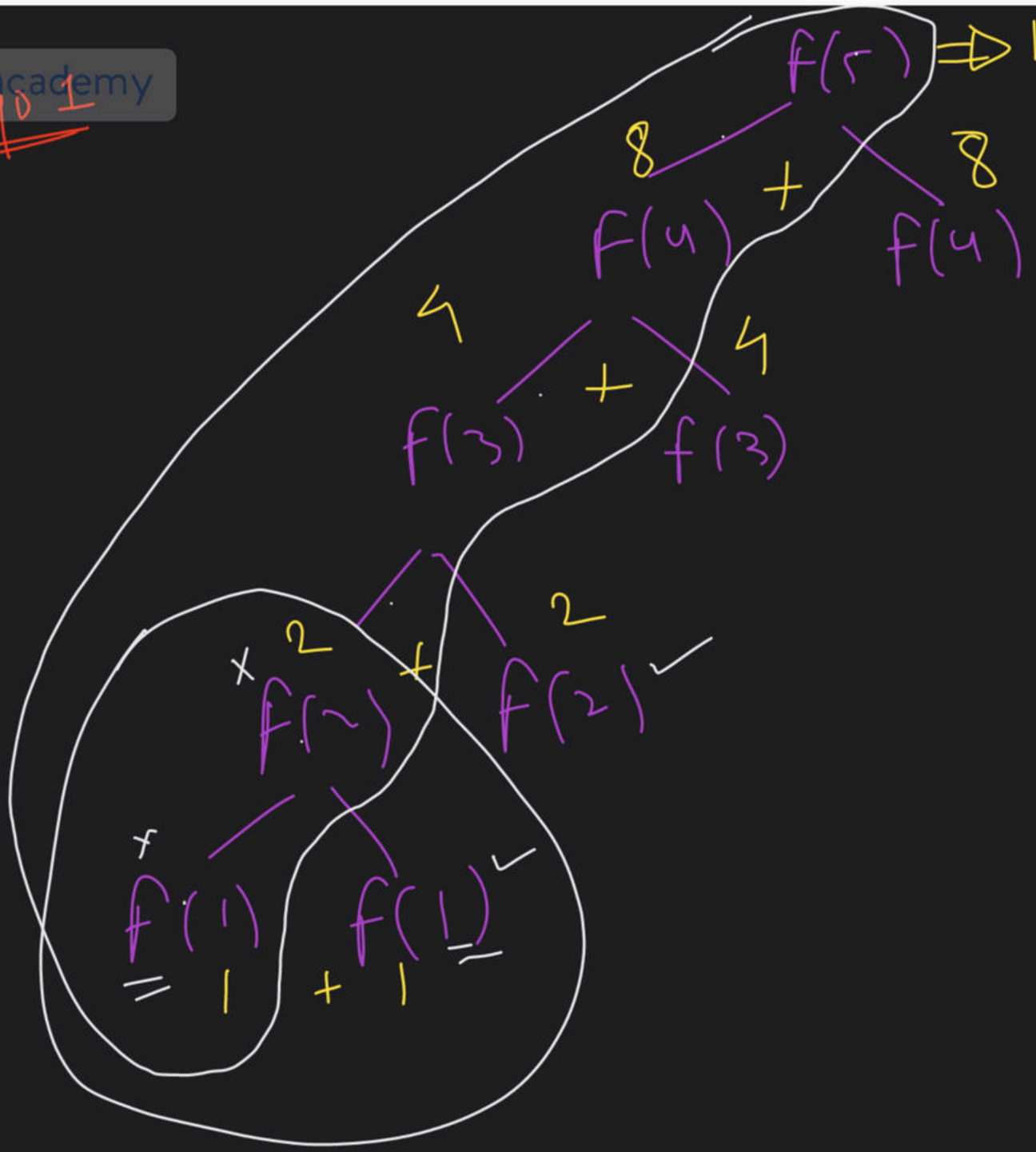
```python
def fun(n):
    if n <= 1:
        return n
    else:
        return 2 * fun(n - 1)

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

$T(1) = 1$

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

$\hookrightarrow \Theta(n)$

Algo 1



$\Rightarrow$ Ans (16)

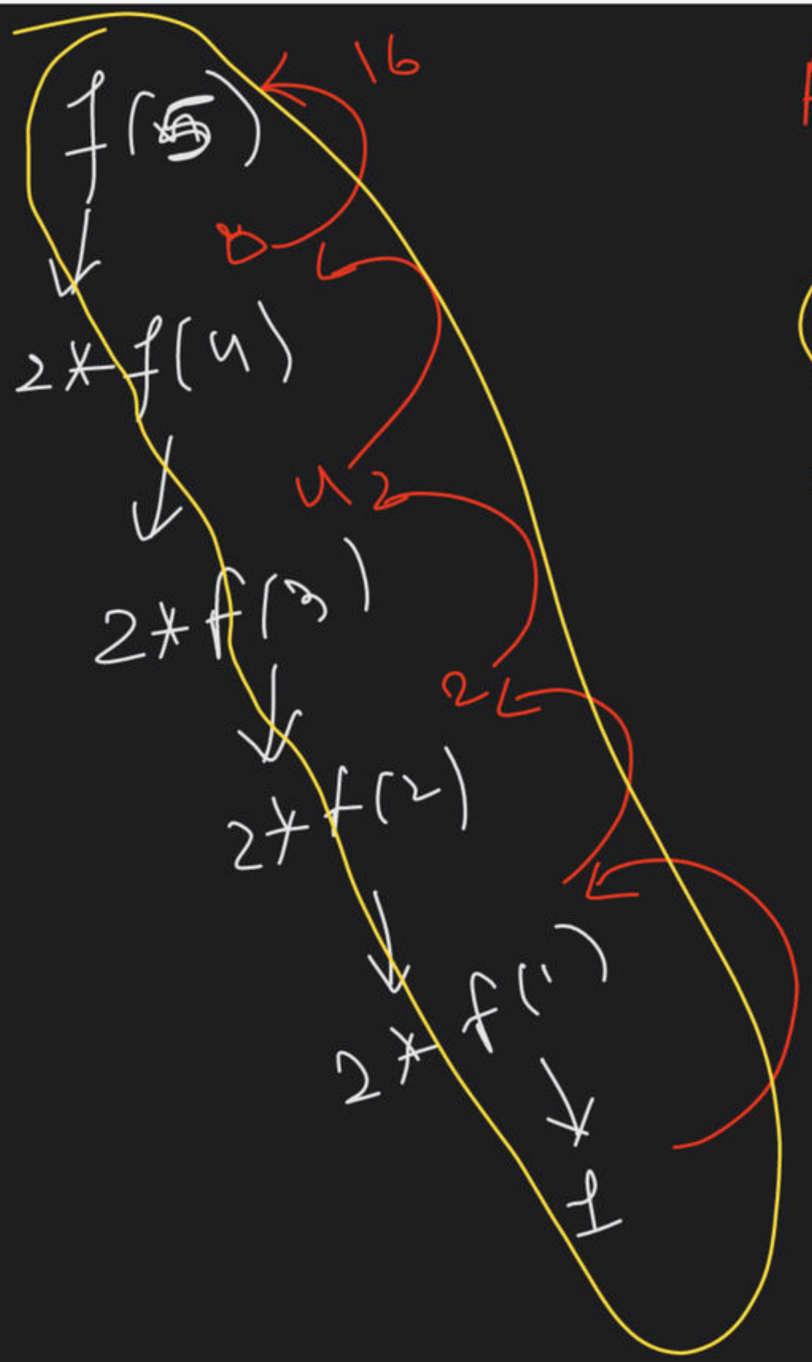
Time $\Rightarrow \underline{\Theta(2^n)}$ ✓

Space $\Rightarrow$ Height of tree = CES

$\Rightarrow n$

Space $\Rightarrow \underline{\Theta(n)}$ ✓

Algo 2



Ans ⇒ 16

Time ⇒ $\Theta(n)$

Space ⇒ $\Theta(n)$

unacademy

C Prog

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

Python

```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)
```

Time =

Space =

$T(1) = 1$

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

$\Rightarrow O(n^2)$

Space $\Rightarrow$ D.S + CES

$1 + n$

$\Rightarrow O(n)$

```
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)
```

Time $= \theta(n)$

Space $= \theta(\log n)$

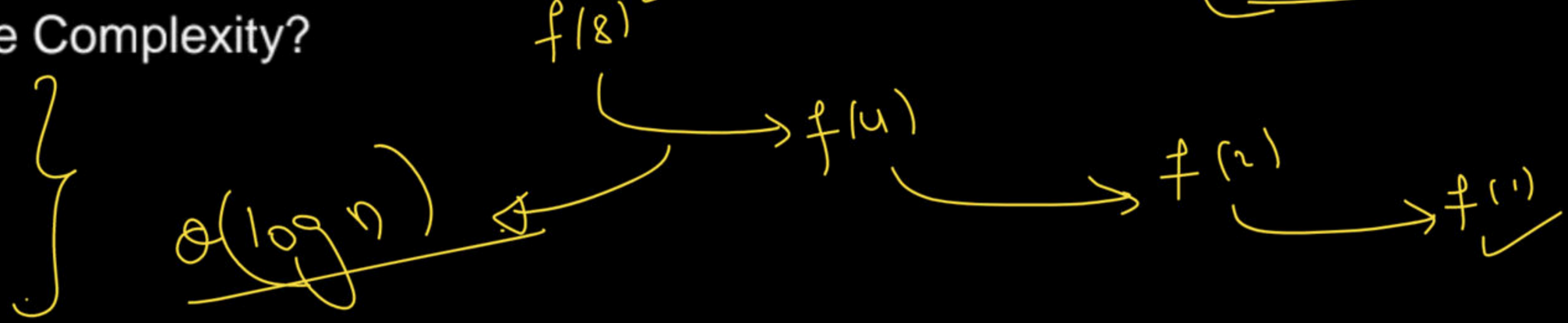
$T(n) = T(n/2) + n$

$\Rightarrow n^0 < n$

$\theta(n)$

What is Space Complexity?

(a) $\boldsymbol{\theta}(n)$
(b) $\boldsymbol{\theta}(\log n)$
(c) $\boldsymbol{\theta}(n \log n)$
(d) None



Sorting techniques

1, 10, 2, 5, 7

↳ 1, 2, 5, 7, 10 (asc)

10, 7, 5, 2, 1 (desc)

Sorting tech.

- Comparision Based
  - (1) Bubble Sort
  - (2) Selection Sort
  - (3) Insertion Sort
  - (4) Quick Sort
  - (5) Merge Sort
  - (6) Heap Sort

  (1), (2), (3) — Subtract & Conquer

  (4), (5), (6) — Divide & Conquer

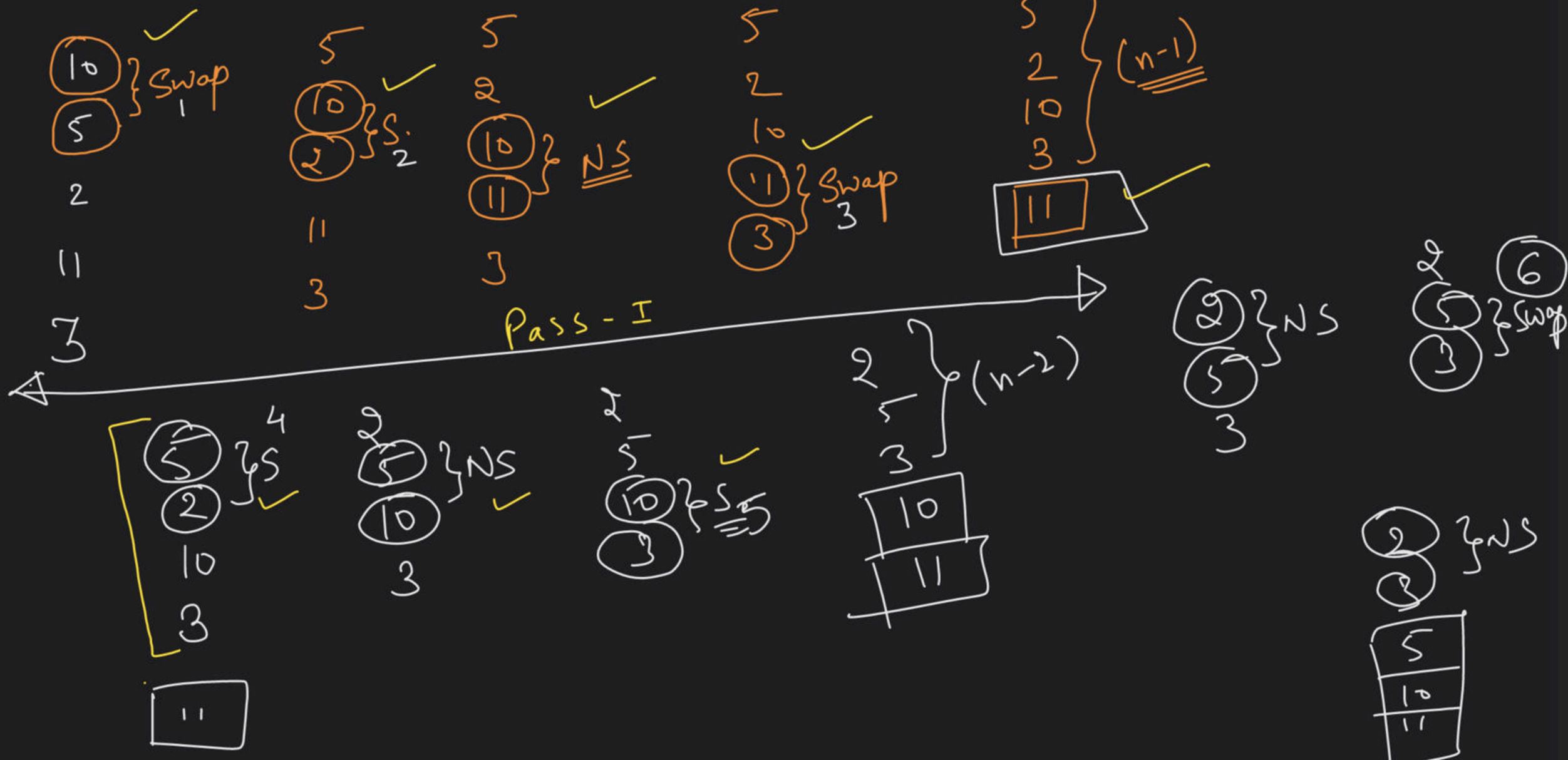
  (5), (6) — tree Based

- Counting Based
  - Radix Sort

Bubble Sort :-

$$(n-1) + (n-2) + (n-3) + \cdots\cdots 1$$

$$\Rightarrow \frac{n(n-1)}{2} \Rightarrow \underline{\Theta(n^2)}$$



unacademy

Complexity

- Best Case $\Rightarrow \Theta(n^2) \Rightarrow \underline{\Theta(n)}$
- Worst Case $\Rightarrow \Theta(n^2)$
- Av. Case $\Rightarrow \Theta(n^2)$

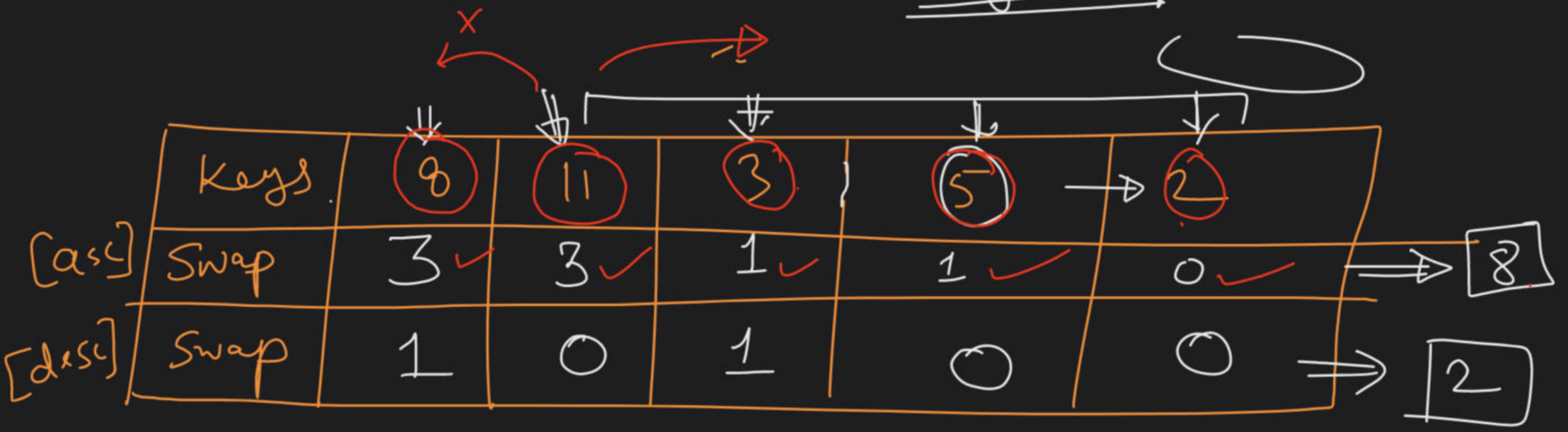
No of swap
- max $\Rightarrow \frac{n(n-1)}{2}$
- Min = 0

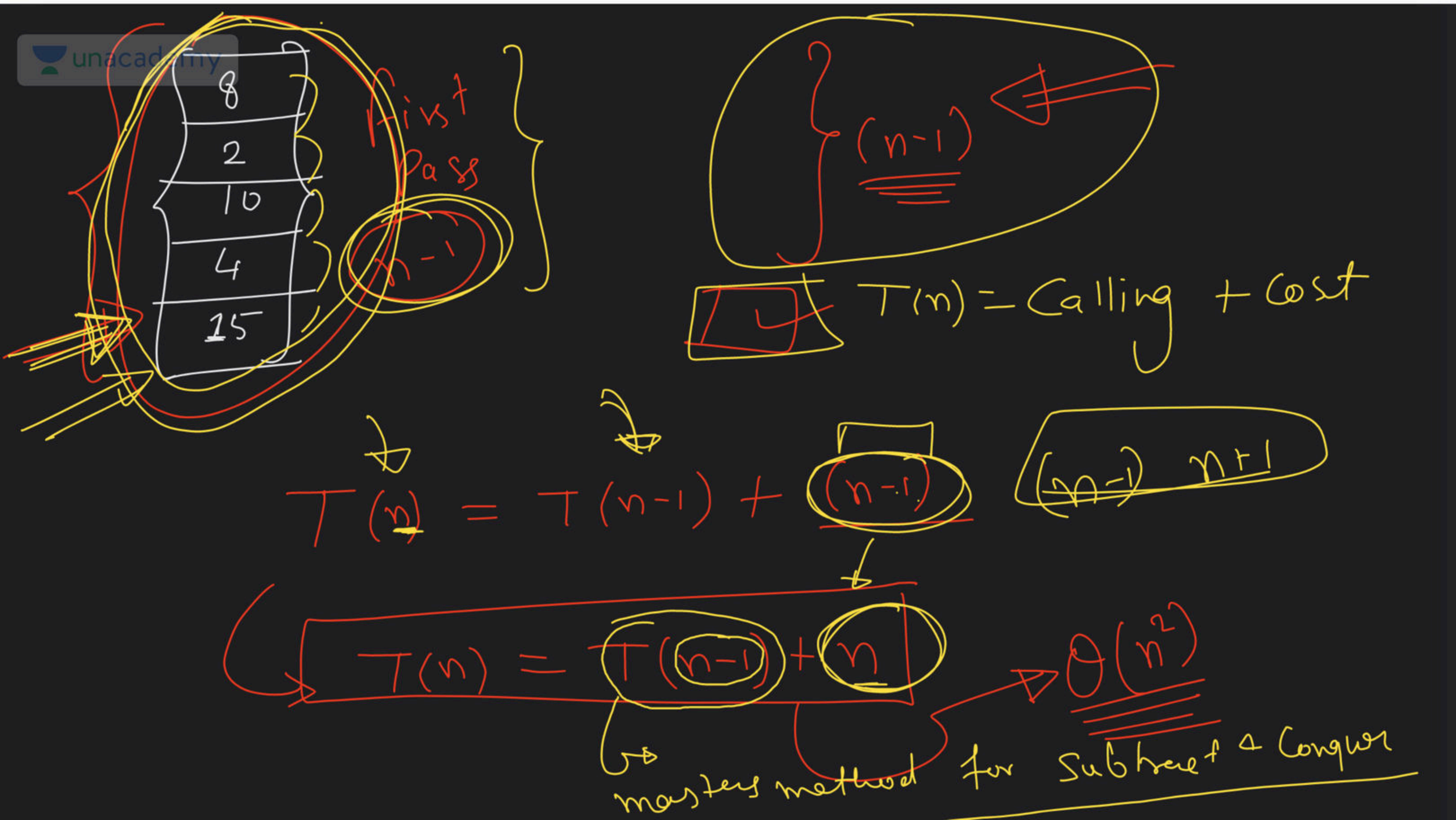
unacademy

8
11
3
5
2

No of Comparisons :→ 10

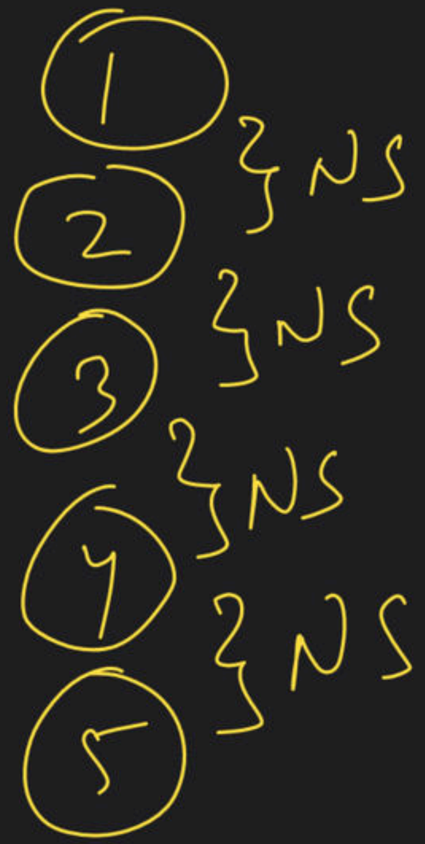
No of Swaps :→




8
2
10
4
15
First Pass
(n-1)
(n-1)
T(n) = Calling + Cost
T(n) = T(n-1) + (n-1)
(n-1) n+1
T(n) = T(n-1) + n
O(n²)
masters method for Subtract & Conquer

$T(n) \Rightarrow T(n+1)$

$T(50) =$ ——

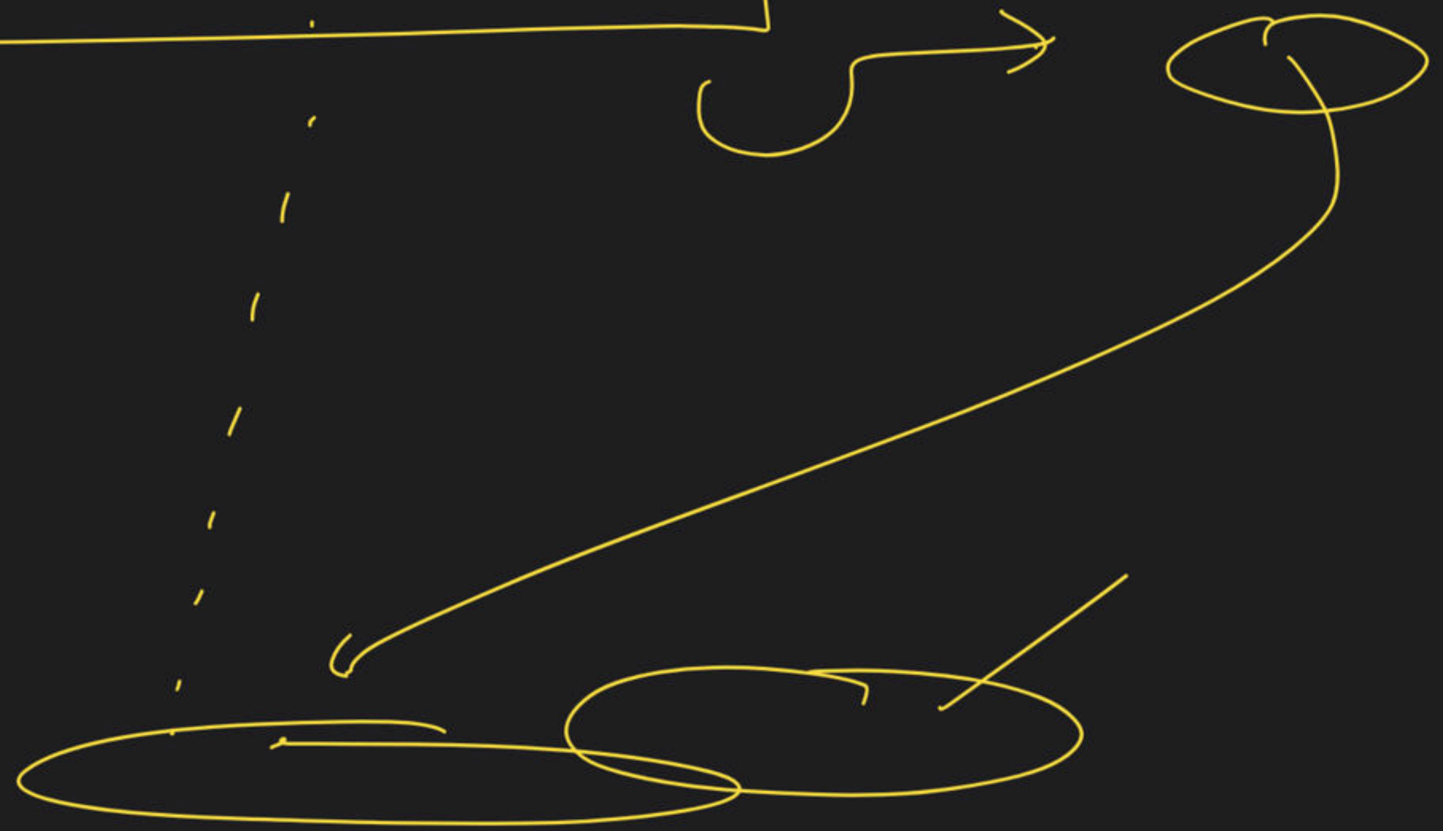


4

n-1

$\rightarrow O(n)$

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



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

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

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

unacademy

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

$a = 1$ —

$a > 1$ —

$a < 1$ —