Array (1D Array / 2D Array)

↓

Problem - Solving

Recursion

→ function is calling itself

(Method definition)

↓ Demonstration

① → factorial of any given number

$5! = 5 * 4 * 3 * 2 * 1$

$5! = 120$

$\{ 0! = 1 \quad 1! = 1$

$n = 1$

↳ return 1

fact(n):

① Base case condition

(Stop)

Recursive code

② Recursive function call

$n > 1$

↳ return

$n * fact(n-1)$

$5! = 5 * 4!$

$4 * 3!$

$3 * 2!$

$0! = 1$ ✓

$n = 0$

$\{ 1^0 = 1 \quad 2^0 = 1 \}$

constraint

$n >= 0$

Method name

```
fact(n):
    if n <= 1:
        return 1
    else return n * fact(n-1)
```

(0,1)

$n = 1$

$1! = 1$

Base case condition

① ②

fact(500)

500 * fact(499)

499 * fact(498)

fact(5) 120

5 * fact(4) 24

4 * fact(3) 6

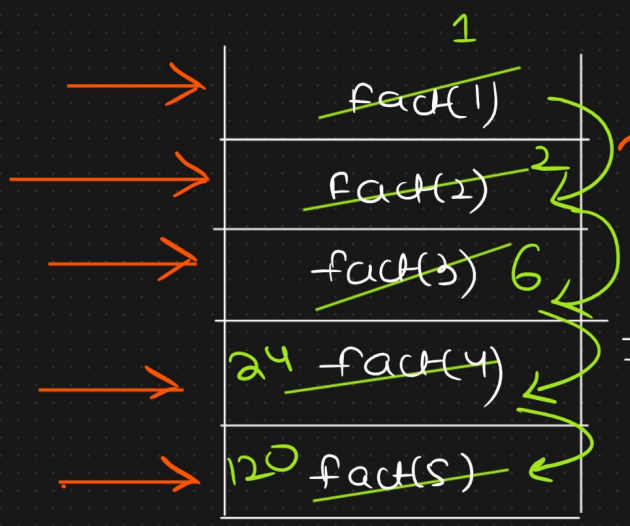
3 * fact(2) 2

2 * fact(1)

1

LIFO (Last In First Out)

Stack Data Structure



Stack

Space complexity

O(n)

Stack

↳ to store the function call

Base case Condition

① Base case condition

↳ Directly return the result

② Recursive function call

Time & Space complexity

```
## Factorial of a given number
## Method definition via the recursion
def factorial(n):
    ## 1. Base case condition
    if n <= 1:
        return 1

    ## 2. Recursive function call
    return n * factorial(n-1)

## Driver code
n = 50
result = factorial(n)
print(result)
```

T(n)

Base case condition — C

T(n-1) + C

Recurrence Relation

constant — O(1)

$$T(n) = \begin{cases} C & ; n <= 1 \\ T(n-1) + C & ; n > 1 \end{cases}$$

Recursive function call

$T(n) = T(n-1) + C$ — $O(n)$

Linear Time Complexity

for every Recursive code

↓

Recurrence Relation

(solve)

↓

Big O time complexity

1. Substitution Method
2. Recursive Tree Approach
3. Master's Theorem

(2) Fibonacci Series

n = 5 / 8

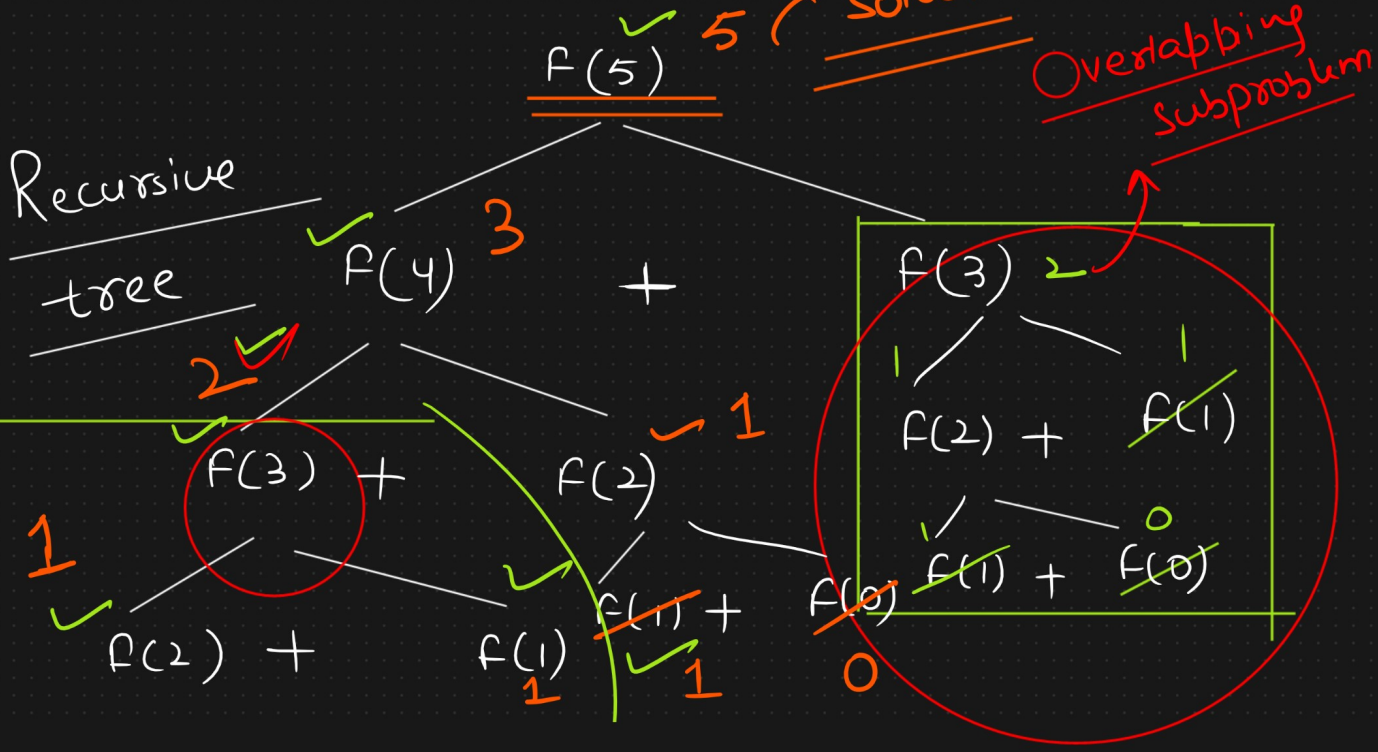
output = 5 / 21

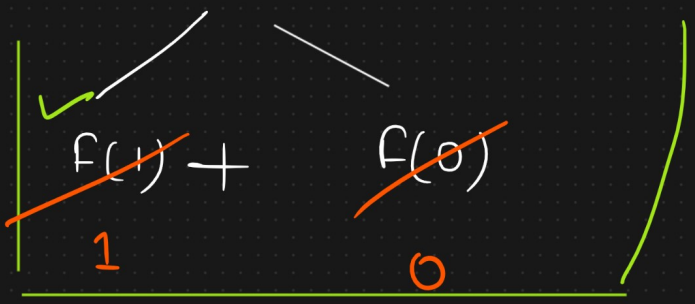
0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	5	8	13	21	34

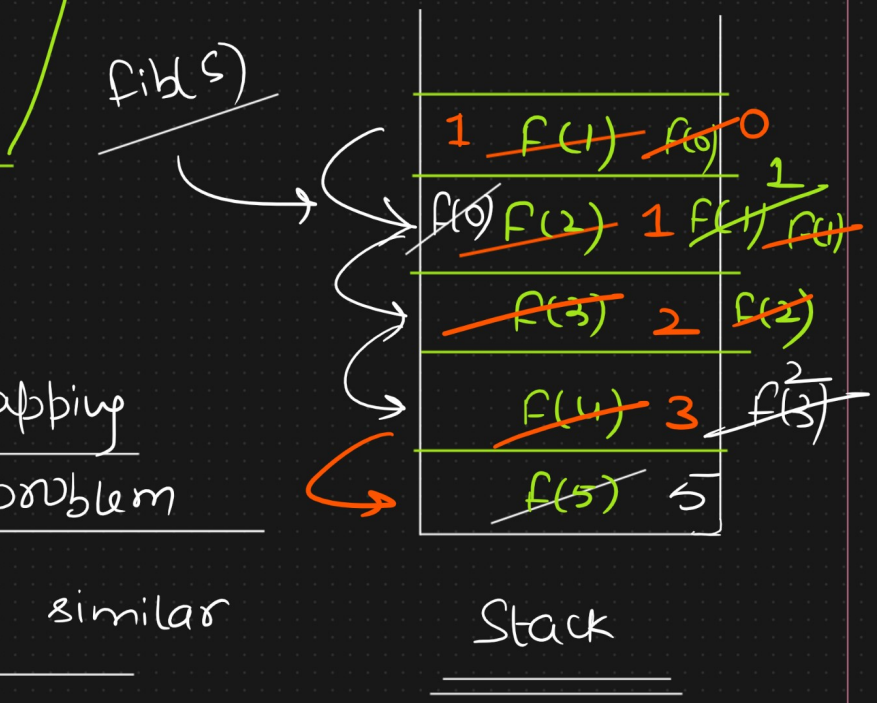
Method name

```
fibonacci(n):
    if n <= 1:
        return n          } (1)
    return fibonacci(n-1) +
           fibonacci(n-2)
```

(2)







Overlapping Subproblem

Re-computation of similar function calls

Wastage of time

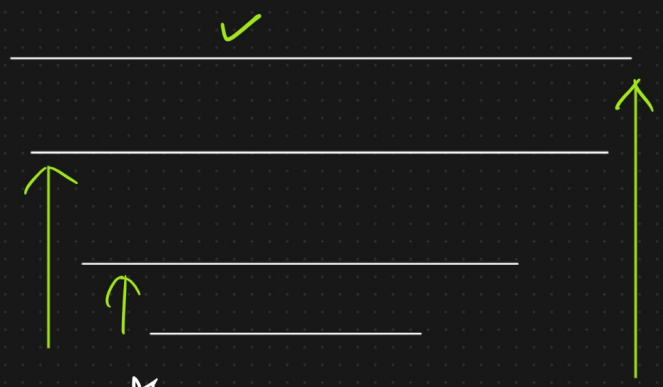
n is very big

More execution time

Time Limit Exceeded

Time complexity ↑↑

Climbing Stairs



Constraints

Jump 1 step at a time

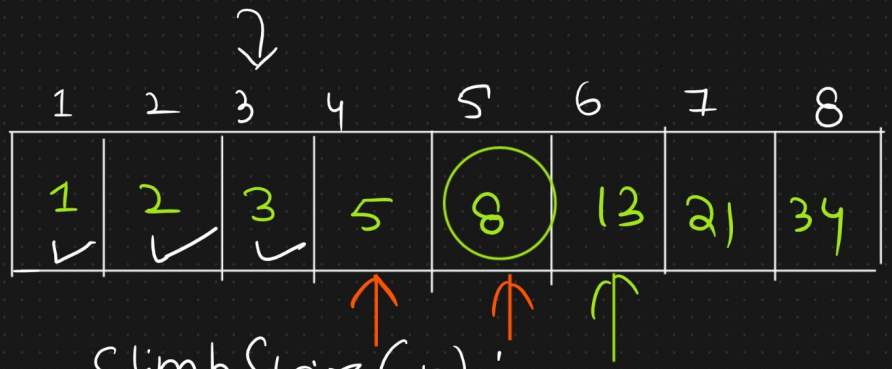
2 step at a time

$n = 1$ → 1 way

$n = 2$ → 1, 1 ; 2 → 2 ways

$n = 3$ → 1, 1, 1 ; 1, 2 ; 2, 1 → 3 ways

$n = 4$ → 1, 1, 1, 1 ; 1, 2, 1 ; 2, 1, 1 ; 1, 1, 2 ; 2, 2 → 5 ways



climbStairs(n):

if (n <= 3):

return n

return climbStairs(n-1) + climbStairs(n-2)

```
class Solution:
    def fib(self, n: int) -> int:
        ## base case condition
        if n <= 1:
            return n

        ## recursive function call
        return self.fib(n-1) + self.fib(n-2)
```

T(n)

1st statement → C

Base case condition

n > 1

T(n-1) + T(n-2) + C

Fibonacci Series

Recurrence Relation

$$T(n) = \begin{cases} c & n <= 1 \\ T(n-1) + T(n-2) + c & n > 1 \end{cases}$$

Climbing stairs $n <= 3$

$n > 3$

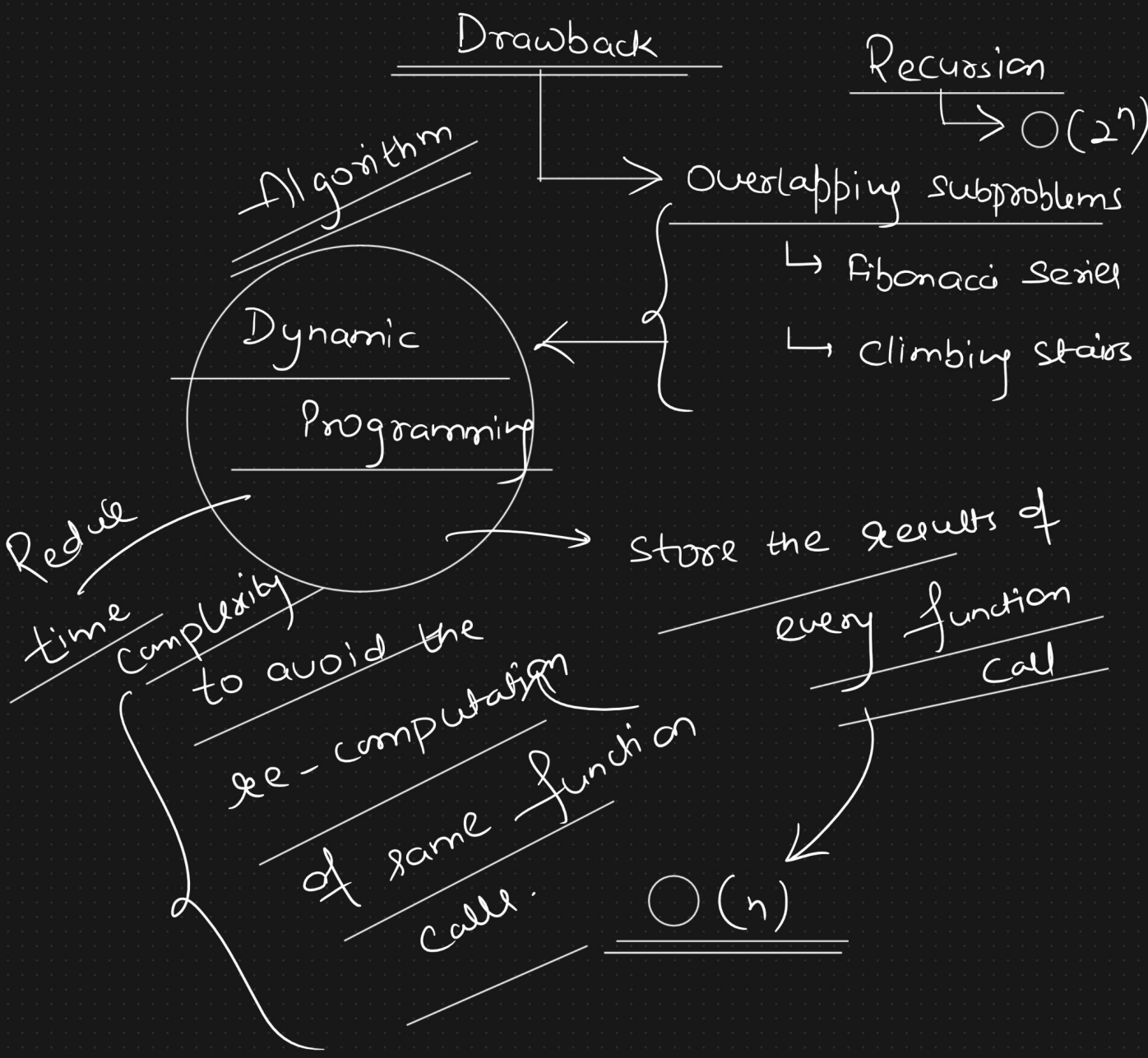
Recursive tree approach → Evaluate

$O(2^n)$

- n is very big → $2^n \approx$ very high → higher execution time
- Exponential time complexity

Space complexity

- Stack → $O(n)$


Drawback
Recursion
O(2^n)
Algorithm
Overlapping subproblems
Fibonacci Series
Climbing stairs
Dynamic
Programming
Reduce
time
complexity
to avoid the
re-computation
of same function
call.
Store the results of
every function
call
O(n)

Most optimized Approach

```
ClimbStairs(n):
    if(n==1)
        return 1;
    }
```

Space complexity = O(1)

time complexity = O(n)

ClimbStairs(5)

8

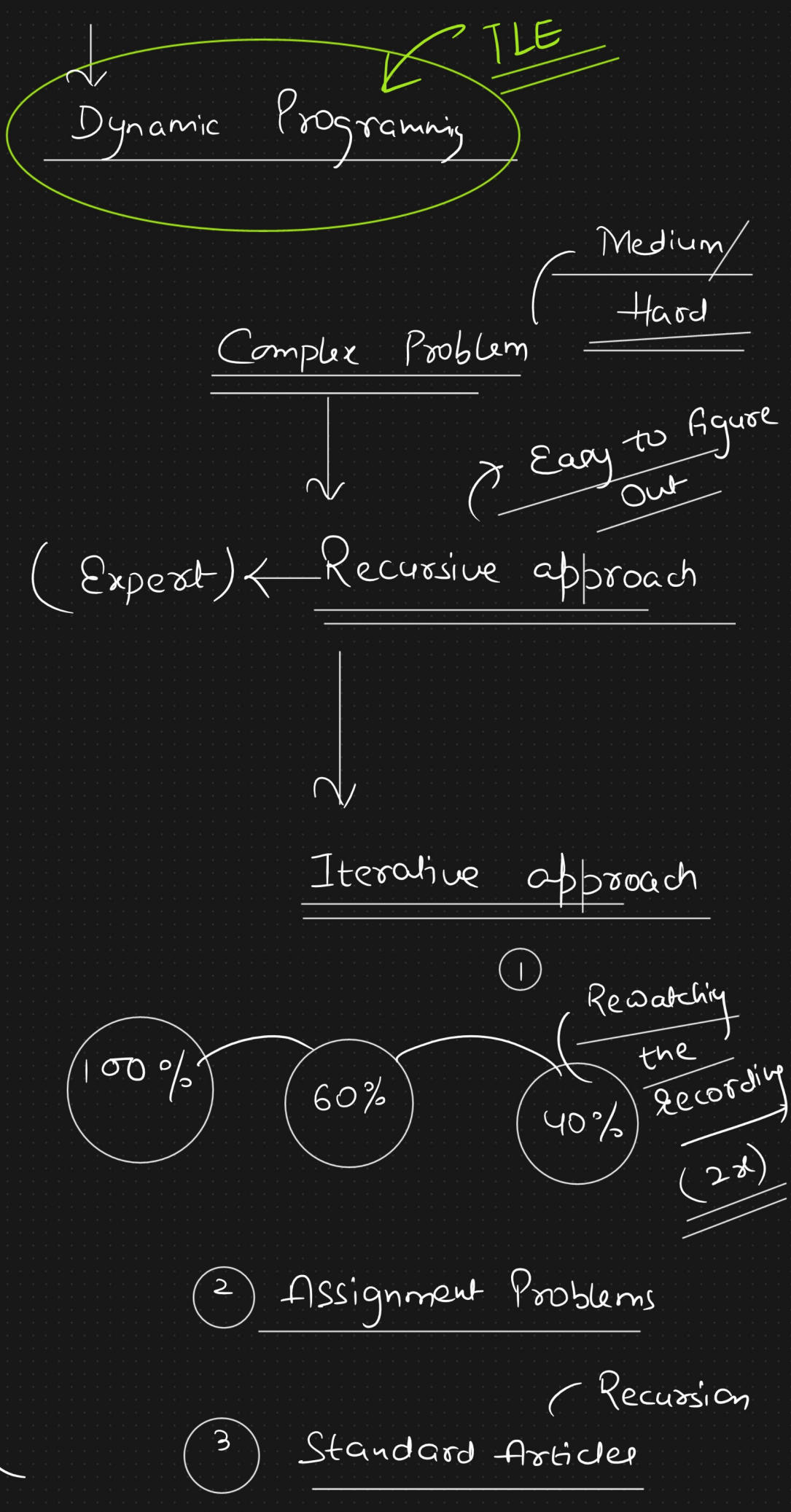
first = 1 1 2 3 5

second = 2 3 5 8

third = 3 5 8

```
for i in range(3, n+1):
    third = first + second
    first = second;
    second = third;
return second
```

	Recursive	Iterative
1	for every recursive code, we have an equivalent iterative solution.	→ loops (for, while)
2	Lesser lines of code → understandable, clarity, reusability	More lines of code
3	Extra space O(n) → Stack space to store function calls	No as such extra space
4	Overlapping subproblem → $O(2^n)$	No as such problem


TLE
Dynamic Programming
Medium/
Hard
Complex Problem
Easy to figure
out
(Expert) ← Recursive approach
Iterative approach
1
Rewatching
the
recording
(2x)
100%
60%
40%
2 Assignment Problems
Recursion
3 Standard Articles

- Recurrence Relation
- Power of any number (Interview)
- Sum of digits & many more problems
- Towers of Hanoi
- Binary Search