* Important Coding Question :→

Balanced Parenthesis :→

string s1 = [()]{}{[()]}

st <char>

Balanced

string s2 = [(])

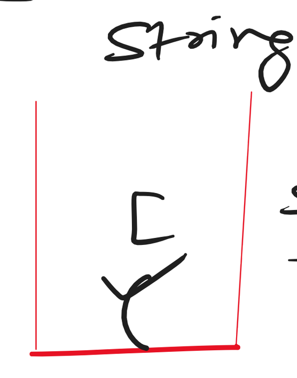
Not Balanced

top = '('

s[i] = { , (, [

( ) [ ]

( )



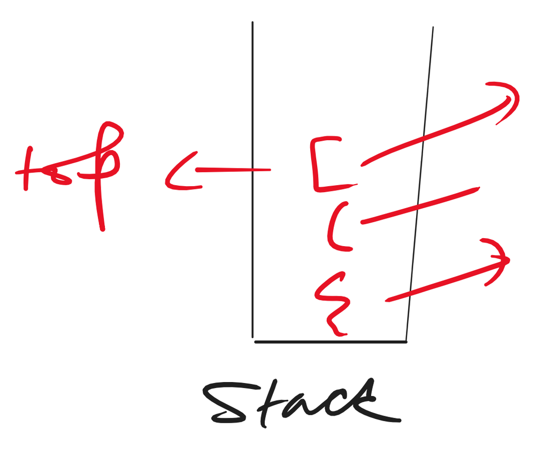
Dry Run → s = {([])}

0 1 2 3 4 5

{ , [, (

↓

push(stack)



pop

pop

pop

Stack / Array Coding Question :
TCS / Accenture / Capgemini

NGE → Next Greater Element

arr[i]

{4, 5, 2, 25}

1 2 3

O/P

4 --> 5

5 --> 25

2 --> 25

25 --> -1

4 --> arr[i]

7, 4, 9, 2, 6, 13

7 → 9

4 → 9

9 → 13

2 → 6

6 → 13

13 → -1

Recursion Dynamic Programming

Those who forget the past are forced to "repeat it."

→ Solving smaller overlapping subproblems to make it easier to solve bigger problems. → DP

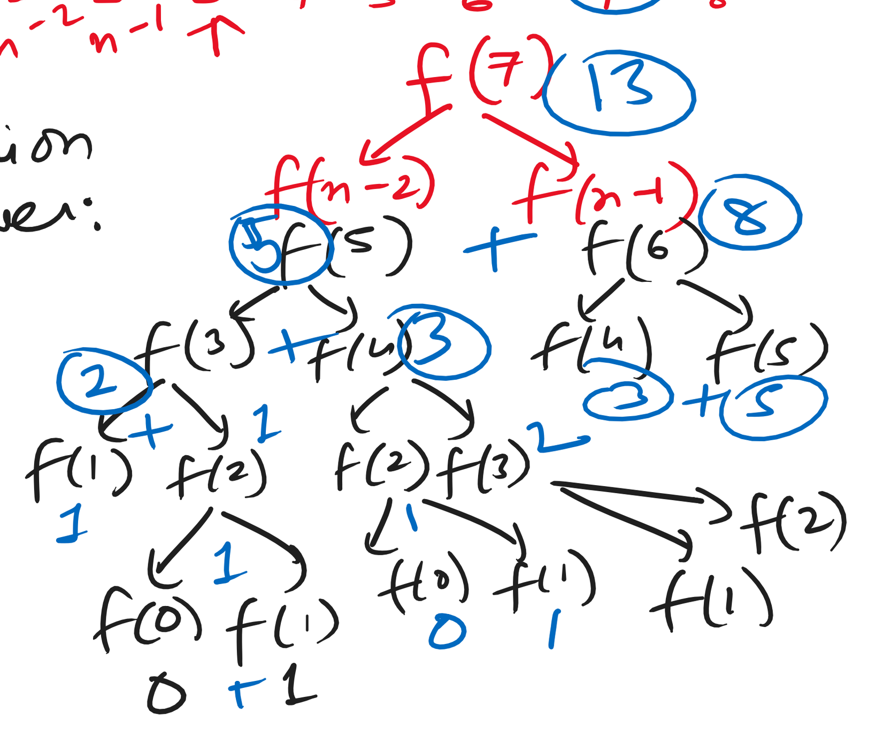
Fibonacci of nth Term (Recursion Tree)

0, 1, 1, 2, 3, 5, 8, 13, 21, - - - - - -

0 1 2 3 4 5 6 7 8

n-2 n-1

Recursion slower:



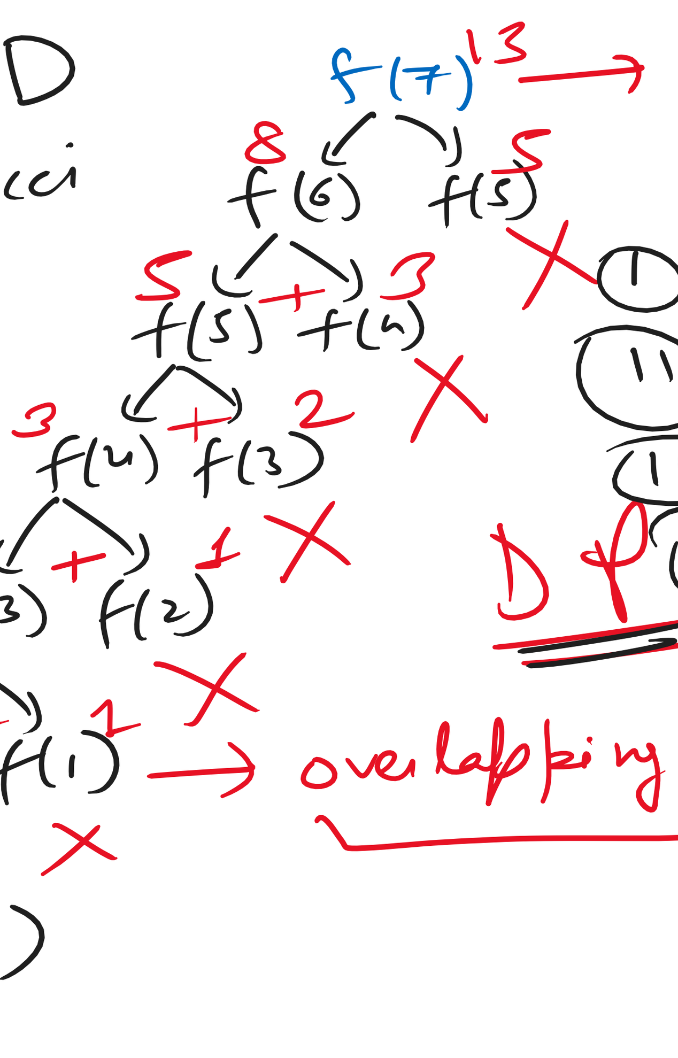
1D, 2D

Fibonacci

LCS

BS + Rec

TLE



→ Bigger problem

I Recursion

II Memoisation

III Tabulation

DP IV SO

→ overlapping subproblem

When you do not calculate the known values & return them directly, it's called Memoisation.

If you store the previous known values in a dp array & only calculate the unknown values, it is called Tabulation.

dp(n+1) =

0	1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6

n = 6

for (i = 2; i <= n) (Tabulation) known values 2 to n

When we use an array, the space complexity of the code becomes O(n) for n elements & it is not memory efficient. Therefore, if possible we should try to only use some variables to solve our problems. When we use variables the Space Complexity of the code becomes O(1) → Constant & the code becomes more efficient and optimized.

[More Optimized Code = Higher Salary]

0 1 2 n

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

p2 p1 curr

3 variables

curr = p2 + p1

p2 = p1

p1 = curr

p2 = 0

p1 = 1

SC = O(1)

Constant

curr = 0 + 1 = 1

p2 = 1

p1 = 1

curr = 1 + 1 = 2

9595