Welcome!

Introduction to Dynamic Programming

Mindset Building

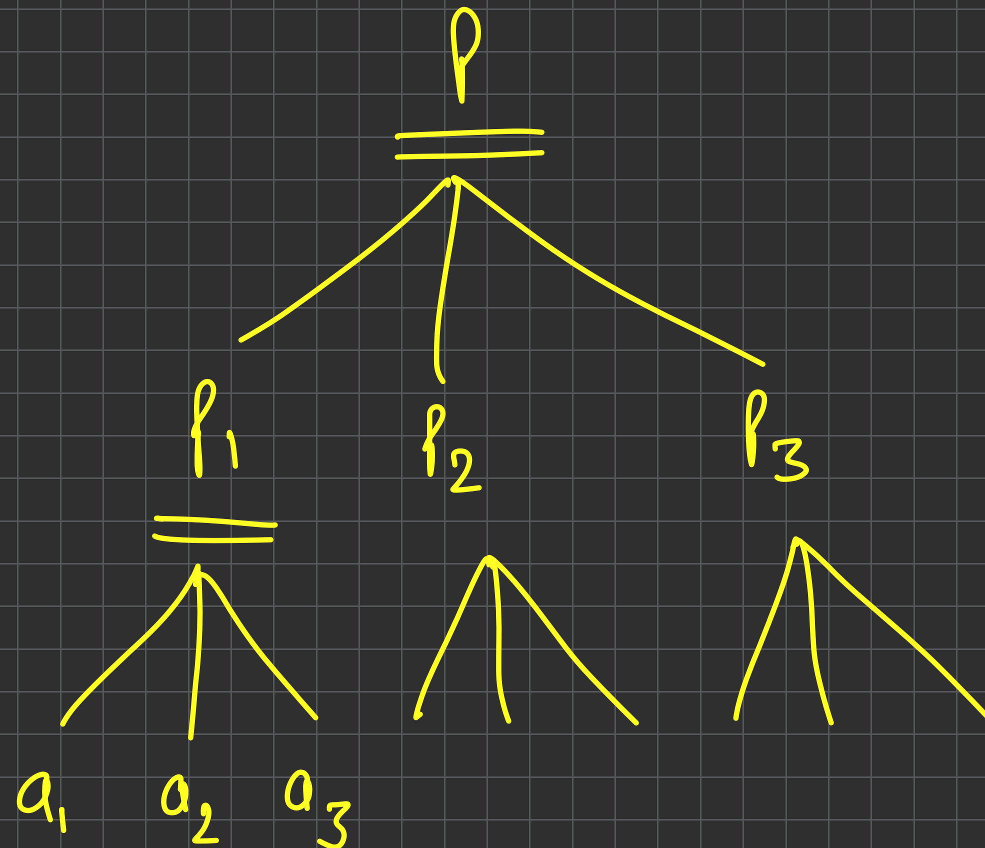
Divide and Conquer mindset

① Breaking a problem into smaller subproblems

② How to get answer for bigger problem from smaller subproblems

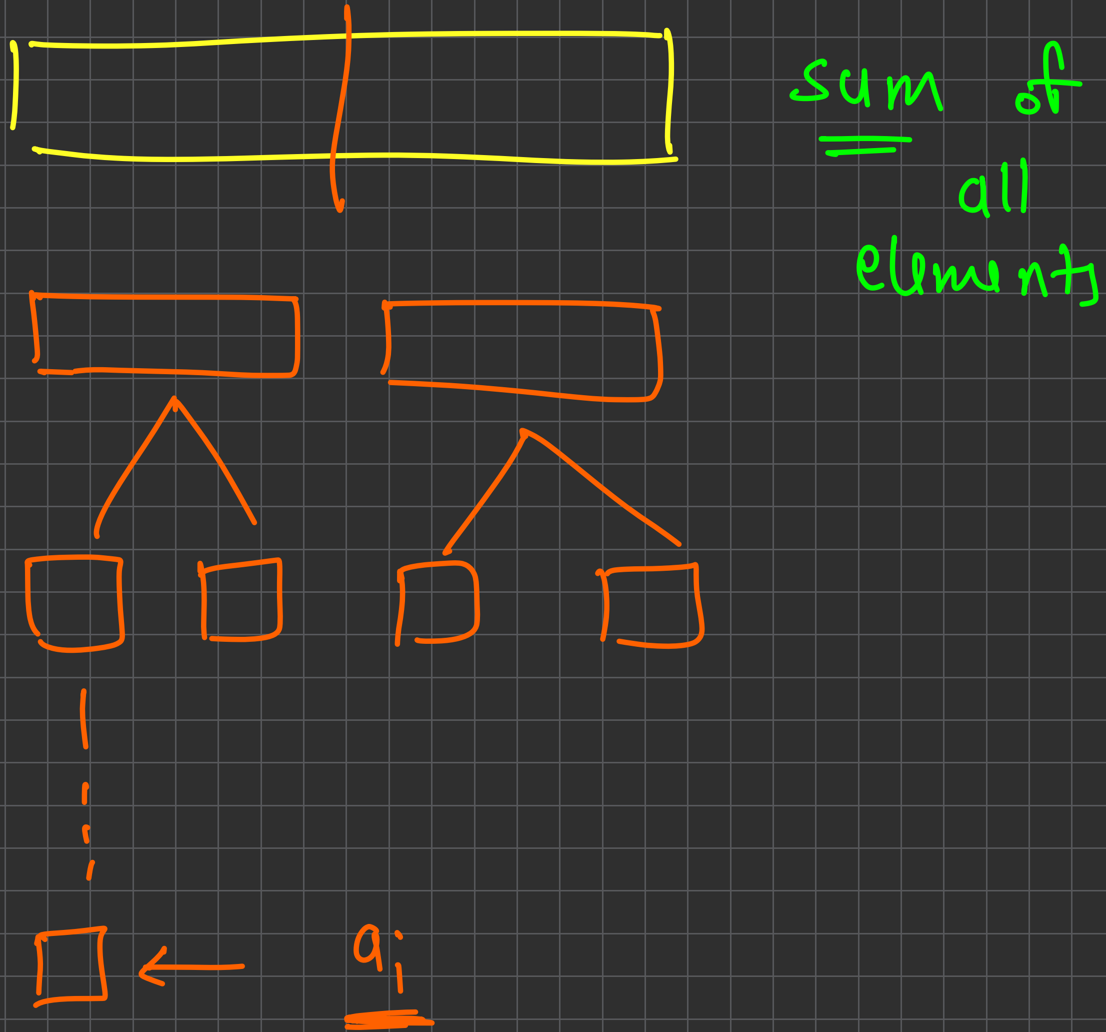
③ What is the smallest subproblem which is trivial to solve

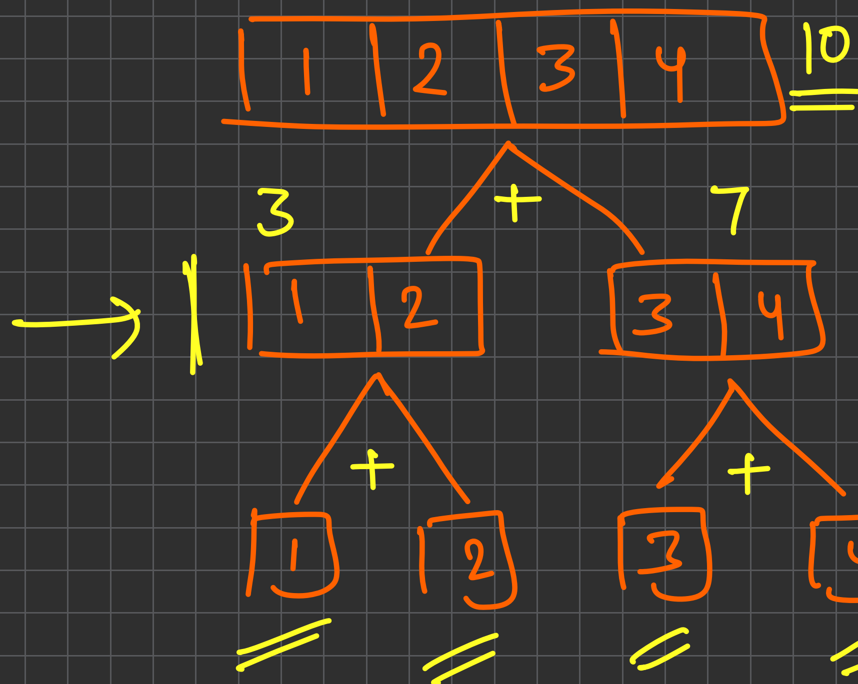
④ What is the biggest subproblem



P
P1
P2
P3
a1
a2
a3

array :

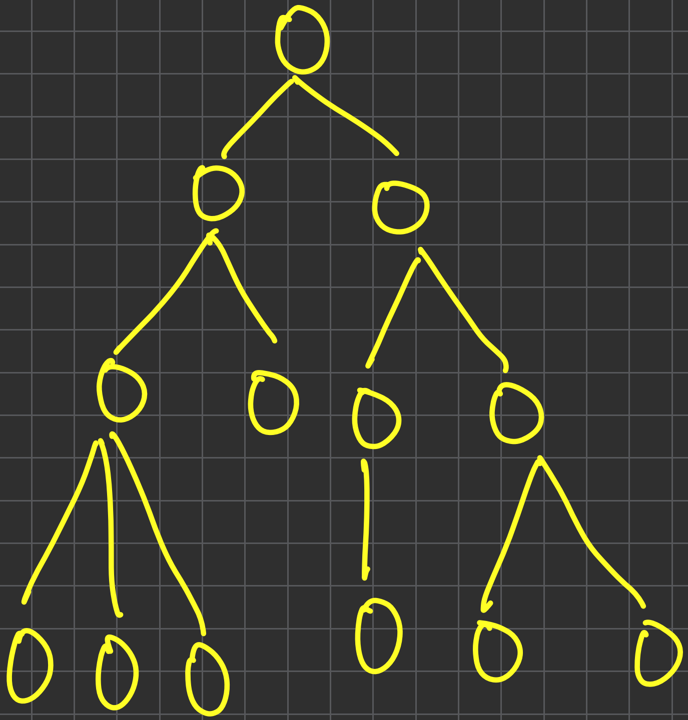




1
2
3
4
10
3
+
7
1
2
3
4
+
+
1
2
3
4

Being clever enough

#storing answer of already calculated problems

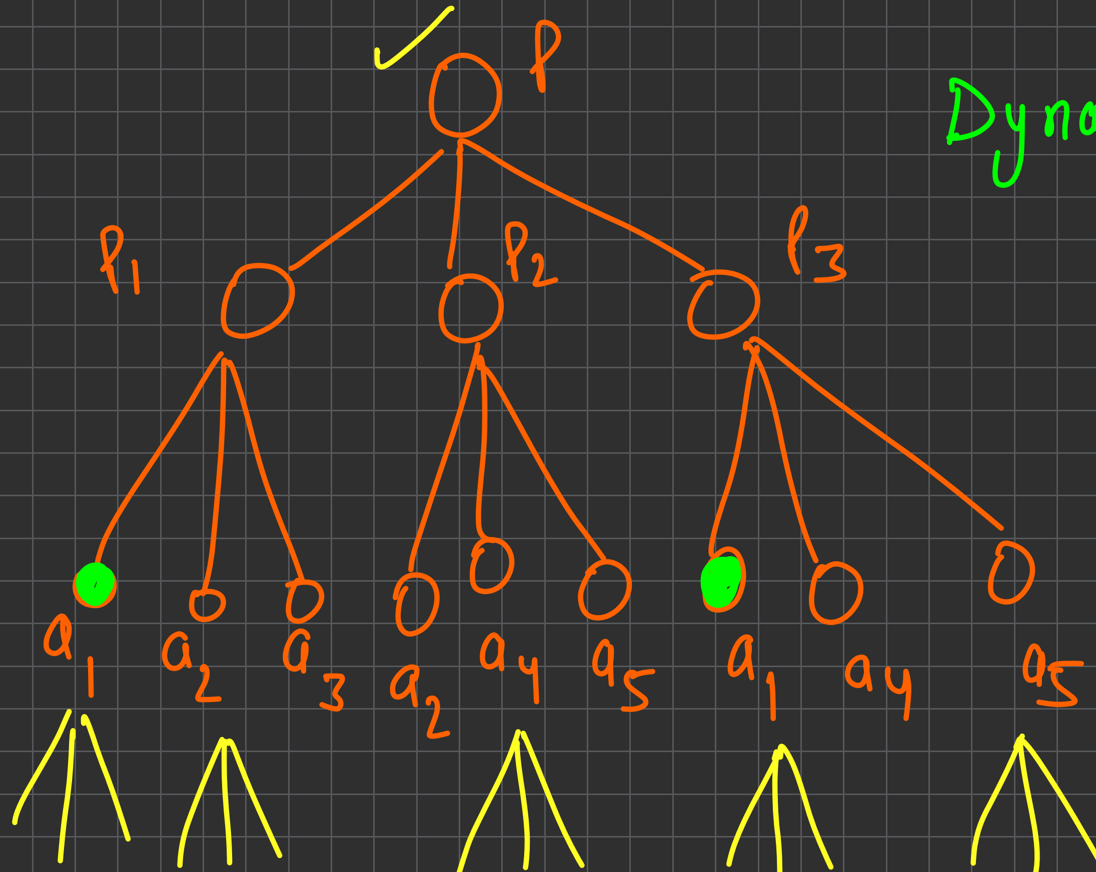


memoization

$a_1 \rightarrow$

$a_2 \rightarrow$

$a_3 \rightarrow$



Dynamic Programming

memoization

Example

fibonacci Numbers $f(n) = f(n-1) + f(n-2)$

$f(1) = 1, f(2) = 1$

nth fibonacci number

$10^5 \geq n \geq 1$

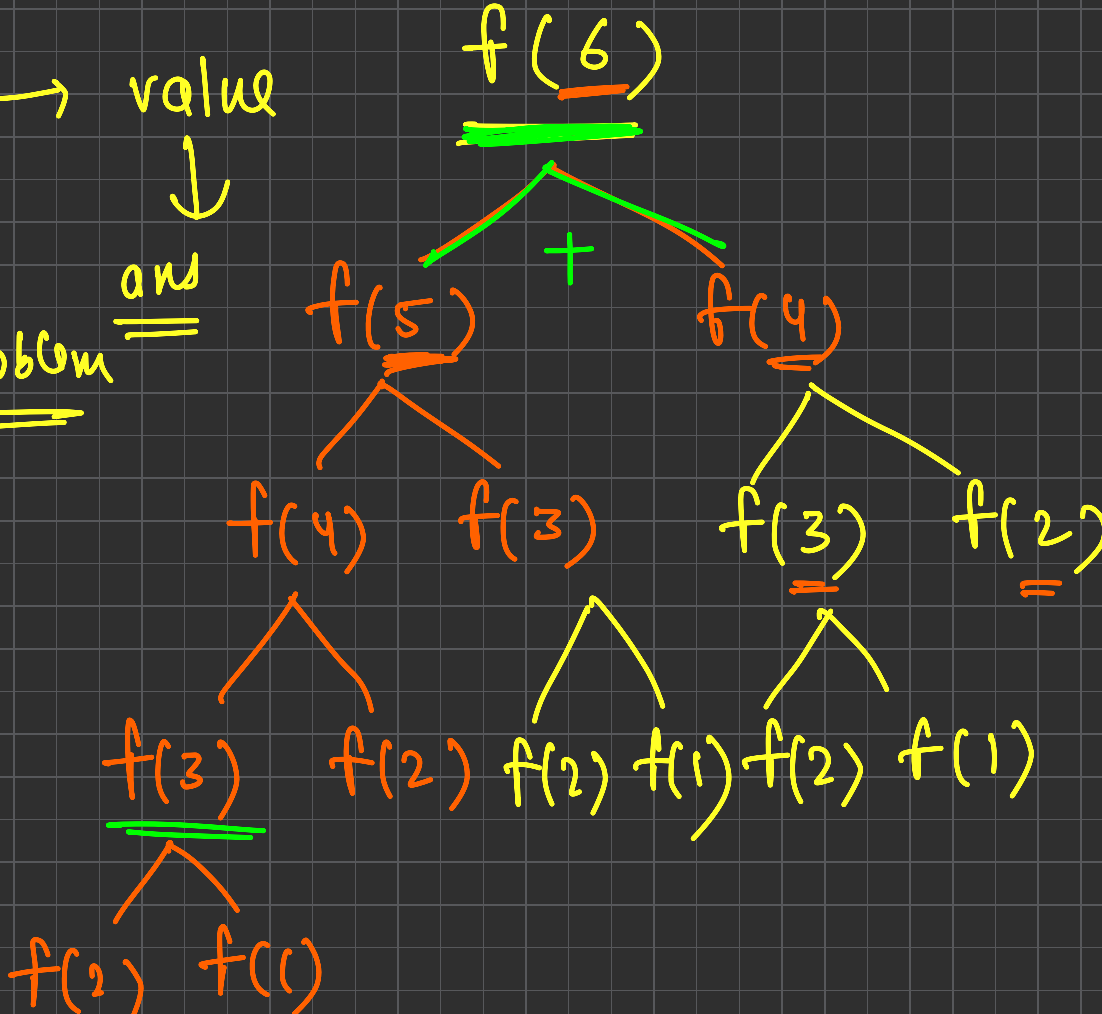
$$f(3) = f(2) + f(1)$$

$$f(5) = f(4) + f(3)$$

key → value

↓ ↓

subproblem ans



Subproblems

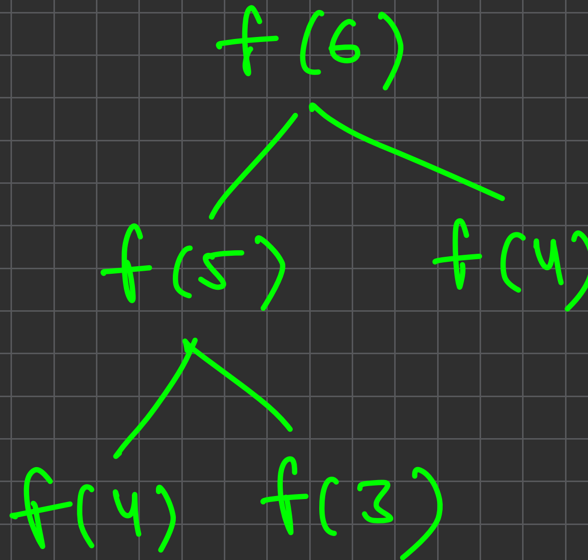
α	1	1	2	3	5	8
0	1	2	3	4	5	6

-10 default value

$1 \leq f(x) \leq 10^{18}$

1. to check if this subproblem has been solved before

2. to update | retrieve the value



$$f(n) \rightarrow f(n-1), f(n-2), f(n-3), f(n-4)$$

$\underline{f(3)}$ after $\underline{f(6)}$

$\underline{f(1)}$, $\underline{f(2)}$, $\underline{f(3)}$, $\underline{f(4)}$ - - - - - -

```cpp
int f(int n){
    if(n <= 2){
        return 1;
    }
    return f(n - 1) + f(n - 2);
}
void solve(){
    int n;
    cin >> n;
    cout << f(n);
}
```

$n = 30$

Brute

not optimal

```cpp
vector<int> dp(100, -1);
int f(int n){
    if(n <= 2){
        return 1;
    }
    if(dp[n] != -1){
        return dp[n];
    }
    dp[n] = f(n - 1) + f(n - 2);
    return dp[n];
}
void solve(){
    int n;
    cin >> n;
    cout << f(n);
}
```

Recursive DP

```cpp
vector<int> dp(100, -1);
void solve(){
    int n;
    cin >> n;
    dp[1] = 1;
    dp[2] = 1;
    for(int i = 3; i <= n; i++){
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    cout << dp[n];
}
```

Iterative DP

$dp(i) \rightarrow dp(i-1),\ dp(i-2)$

How to solve a dp problem?

#Think about subproblem (state) # parameter

#Think about breaking it into smaller subproblems

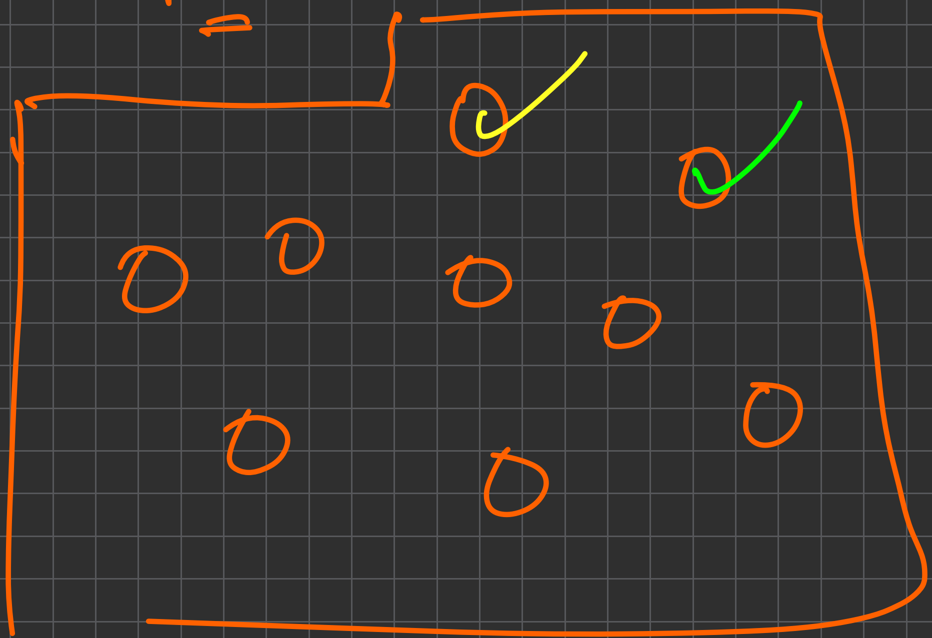
#Think about the relation b/w smaller subproblems to compute bigger subproblem (transition)

#Where does the above relation not work (base case)

#What is the biggest problem to solve (final subproblem)

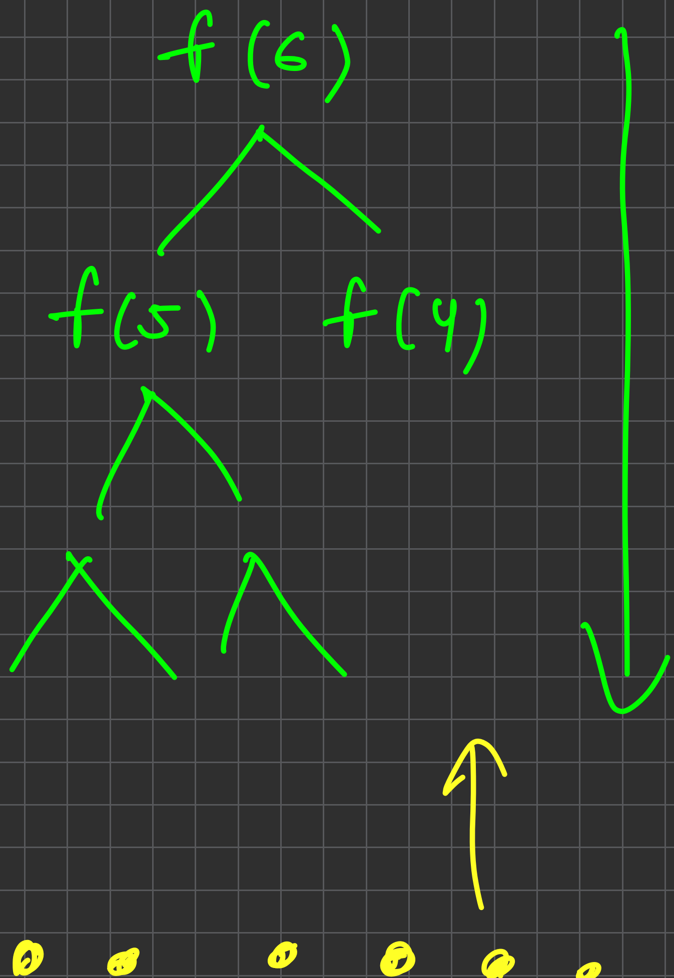
$\boxed{dp(n) = \text{nth fibonocci number}}$

$dp(1)$ different from $dp(5)$



Coding a DP problem

Recursive (top down)	Iterative (bottom up)
Stack space	No stack space
Slower	Faster
Flow is not important (less thinking)	Flow is very important (more thinking)



f(1) f(2) f(3) — . . . . f(6)

bottom up

Time and Space Complexity discussion

tight bound

O(total transition time)

$$\sum_{i=1}^{n} \text{transition time for state } i$$

loose bound

O(no. of states × avg transition time per state)

worst case transition time

fibonacci problem

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

$f(1)$

$f(2)$

$\vdots$

$f(n)$

n states . $O(1)$

$O(n)$

$f(1) \longrightarrow O(n)$

$f(2) \longrightarrow O(n/2)$

$\vdots$

$f(n) \longrightarrow O(n/n)$

n^2

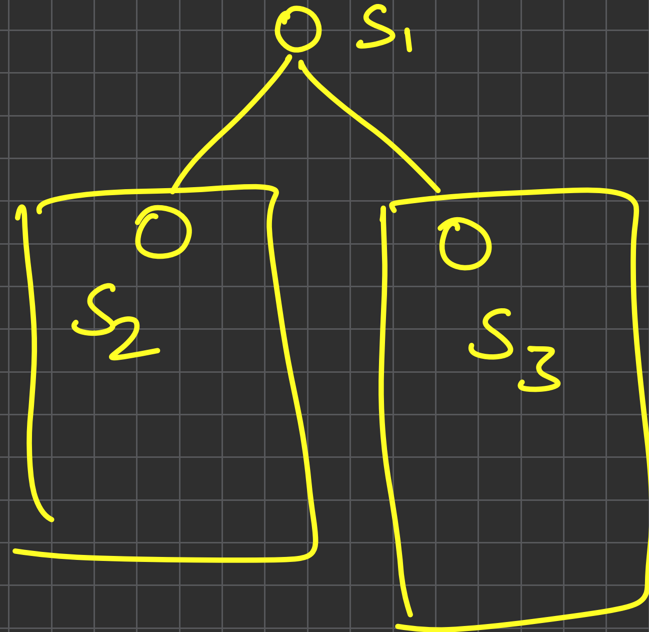
$n \log n$

S_1 O

S_2 O

S_3 O

S_4 O



$$S_1 = 2S_2 + 3S_3 \Big\} O(1)$$

$$S_n \rightarrow aS_1 + bS_2 + cS_3 \cdots xS_{n-1}$$

$O(n)$

L.H.S.

R.H.S

$S_1 \leftarrow$

$S_2 \leftarrow$

$S_3 \leftarrow$

$S_4 \rightarrow S_1 \quad S_2 \quad S_3$

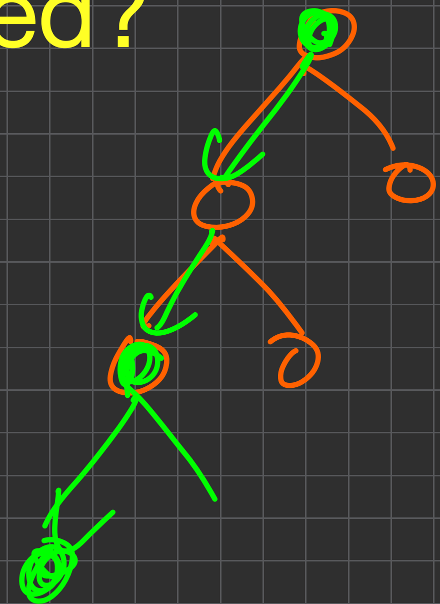
states × avg time required to calculate a state

Where can dp be applied?

① No. of ways

② Min/max answer

③ Checking for possibility



#optimized brute force

#identifying overlapping subproblems