Dynamic Programming

→ "Those who can't remember their past are condemned to repeat it"

① Recursion
- → Time Complexity Poor
- → Space complexity Poor

② Memoization
- → Time Complexity Good
- → Space Complexity Poor

③ Tabulation
- → Time Complexity Good
- → Space Complexity Good

Pre-requisites

Topics

- Dynamic Programming {Level 1 + 2} → Recursion
- Hashmap & Heap {Level 1 + 2}
- Graphs {Level 1 + Level 2} → Generic Tree {DFS, BFS}
- Bit Manipulation → No System → Binary, Decimal
- Array & String → Remaining Ques

19 Apr

Lecture 1 Dynamic Programming {10:30 - 12:00}

→ Fibonacci + Climb Stairs Module

"Those who can't remember their past are condemned to repeat it"

Exponential

Backtracking

TC → ✗

SC → ✗

1. Recursion {Brute force}

$N \le 18$

20

30

Recursive

TC → ✓

SC → ✗

2. Memoization {Top Down DP}

Iterative

TC → ✓

SC → ✓

3. Tabulation {Bottom Up DP}

4. Space Optimization → Limited previous states

Fibonacci Number

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

0^{th} 1^{st} 2^{nd} 3^{rd} 4^{th} 5^{th} 6^{th} 7^{th}

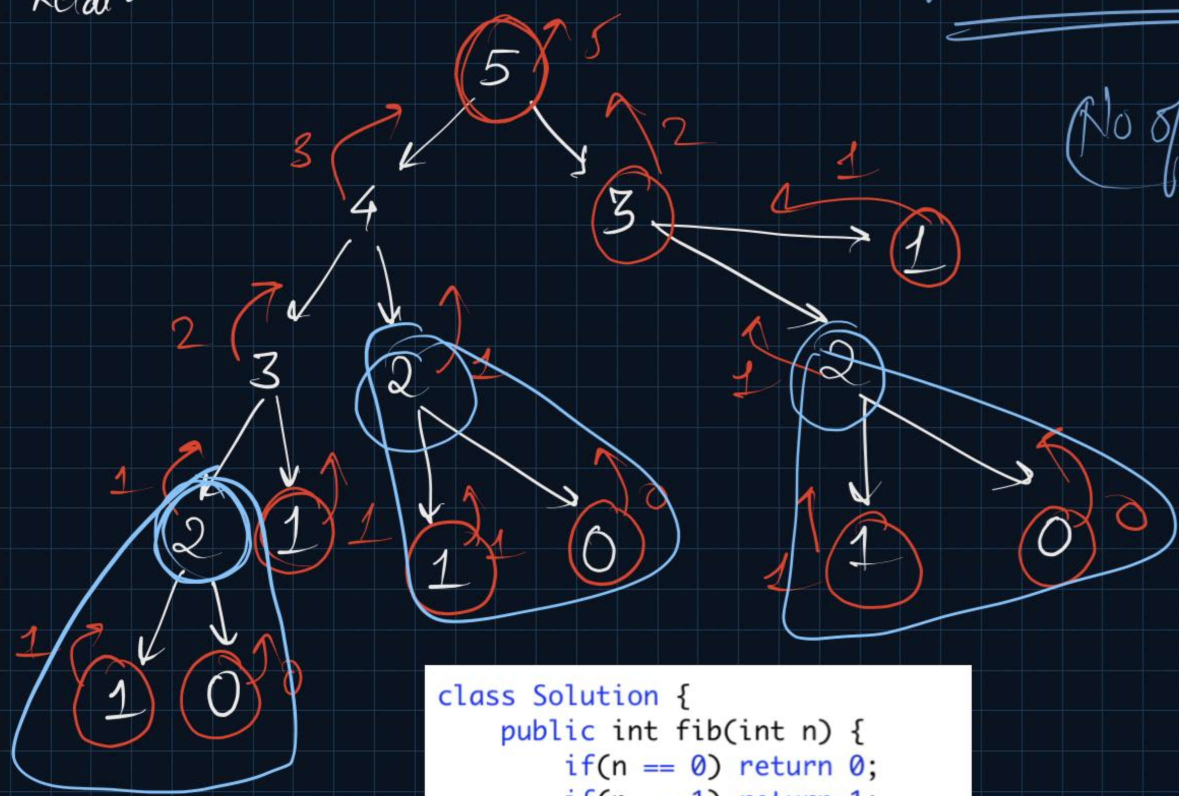
Expectatn :→ N^{th} Fibonacci No. {fib(N)}

Faith :→ fib(N-1), fib(N-2)

$$\boxed{fib(N) = fib(N-1) + fib(N-2)}$$

Recurrence Relatn

Recurrence Relatn $\boxed{fib(N) = fib(N-1) + fib(N-2)}$



Worst case

$(\text{No of calls})^{\text{height}} + (\text{pre} + \text{post}) * \text{height}$

$(2)^N + (k + k) * N$

$\Rightarrow O(2^N)$ Time Complexity

Space Complexity $\Rightarrow O(N)$

R.C.S.S

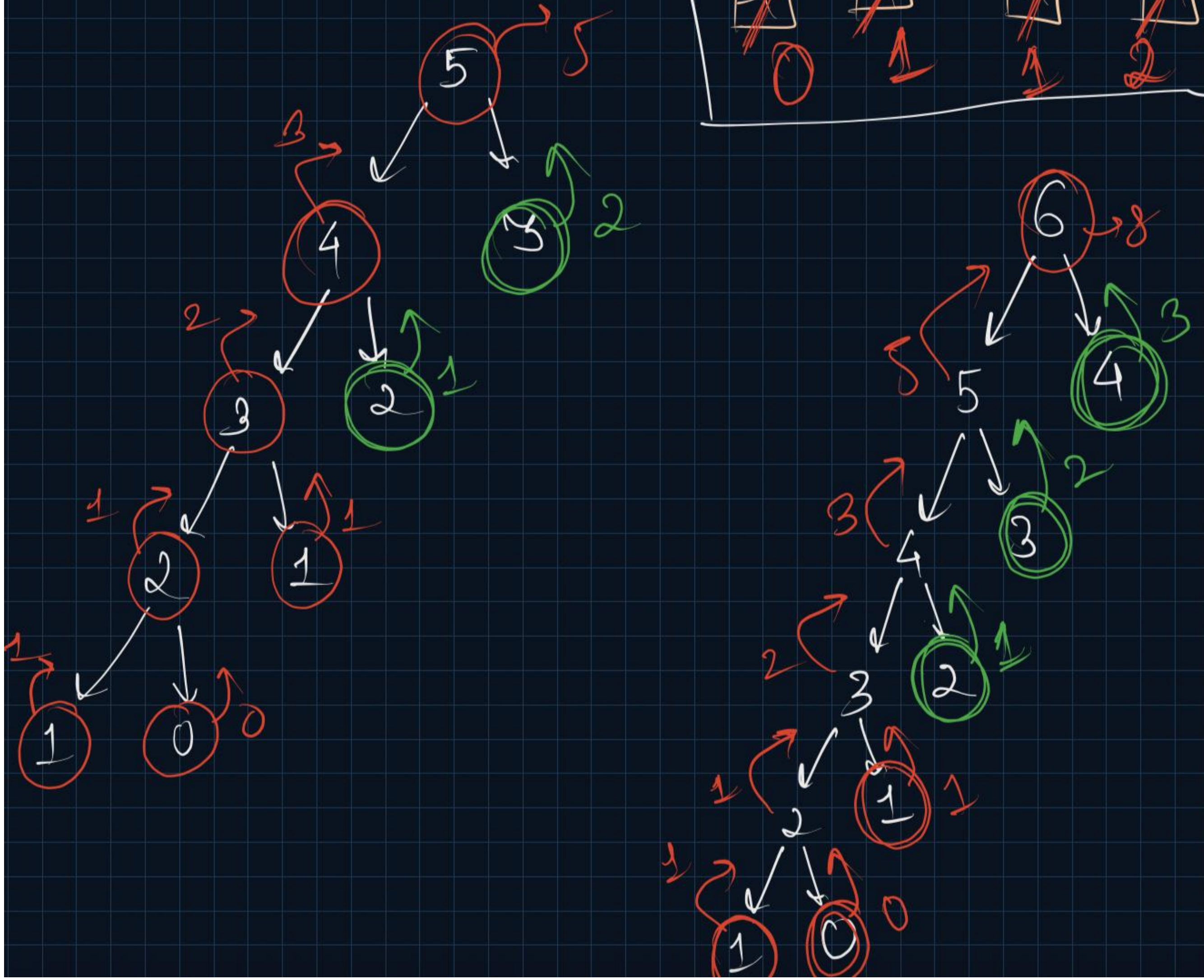
```java
class Solution {
    public int fib(int n) {
        if(n == 0) return 0;
        if(n == 1) return 1;

        int prev1 = fib(n - 1);
        int prev2 = fib(n - 2);

        return prev1 + prev2;
    }
}
```

Memoization

0	1	2	3	4	5	6
~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~
0	1	1	2	3	5	8



O(N) Time Complexity

```java
class Solution {
    public int fib(int n, int[] dp){
        if(n == 0) return 0;
        if(n == 1) return 1;
        if(dp[n] != -1) return dp[n];
        // Already Calculated Value should be returned

        int prev1 = fib(n - 1, dp);
        int prev2 = fib(n - 2, dp);

        dp[n] = prev1 + prev2;
        // Before returning the calculated value, store it somewhere
        return prev1 + prev2;
    }

    public int fib(int n) {
        int[] dp = new int[n + 1];
        Arrays.fill(dp, -1);
        return fib(n, dp);
    }
}
```

Recursion call stack :→ $O(N)$

Extra Space :→ $O(N)$ DP

Time Complexity → $O(N)$

Dynamic Programming Identification

→ 1 Overlapping Subproblems { Repeated calls }

→ 2 Optimal Substructure { Faith }

(3) Tabulation

$$fib(N) = fib(N-1) + fib(N-2)$$

$$DP[N] = DP[N-1] + DP[N-2]$$

(1) $DP[i] \longrightarrow$ Storage & Meaning of the cell

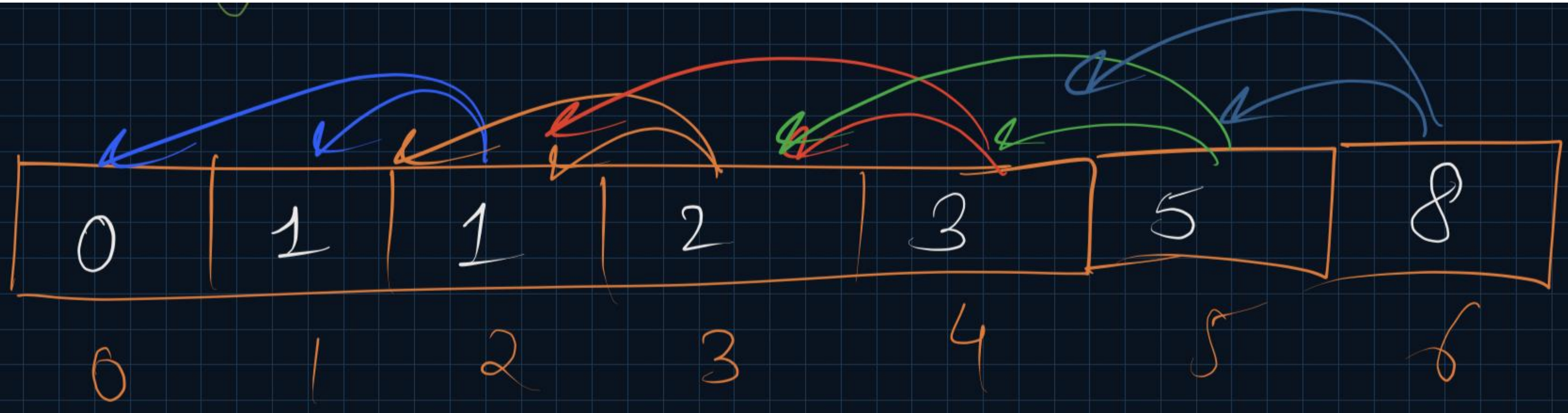
$\longrightarrow$ ith fibonacci No

(2) Smaller Problem $\{ DP[0], DP[1] \}$

$\downarrow$

Bigger Problem $\{ DP[N] \}$

Iterative soln



Smaller $\longrightarrow$ Bigger

```java
class Solution {
    public int fib(int n) {
        if(n <= 1) return n;

        int[] dp = new int[n + 1];
        dp[0] = 0; dp[1] = 1;

        for(int i=2; i<=n; i++){
            dp[i] = dp[i - 1] + dp[i - 2];
        }

        return dp[n];
    }
```

TC $\rightarrow$ O(N)

SC $\rightarrow$ O(N)
(extra space)

R.C.S.S $\rightarrow$ O(1)

4

```
class Solution {
    public int fib(int n) {
        if(n <= 1) return n;

        int prev1 = 0, prev2 = 1;

        for(int i=2; i<=n; i++){
            int curr = prev1 + prev2;
            prev1 = prev2;
            prev2 = curr;
        }

        return prev2;
    }
}
```

Time Complexity → $O(N)$

Space Complexity → $O(1)$