L61
Introduction to Dynamic Programming

Join Discord - https://bit.ly/ly-discord

RECAP

LearnYard

$$\boxed{1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1}\ \ 1\ \ 1\ \ 1\ \ 1$$

$$-\ 10\ +\ 4$$

Let’s take an example
Count number of 1’s

$$\underline{14 \times 12}$$

$$\underline{14 \times 10} + \underline{14 \times 2}$$

Any more real life examples?

Dynamic Programming

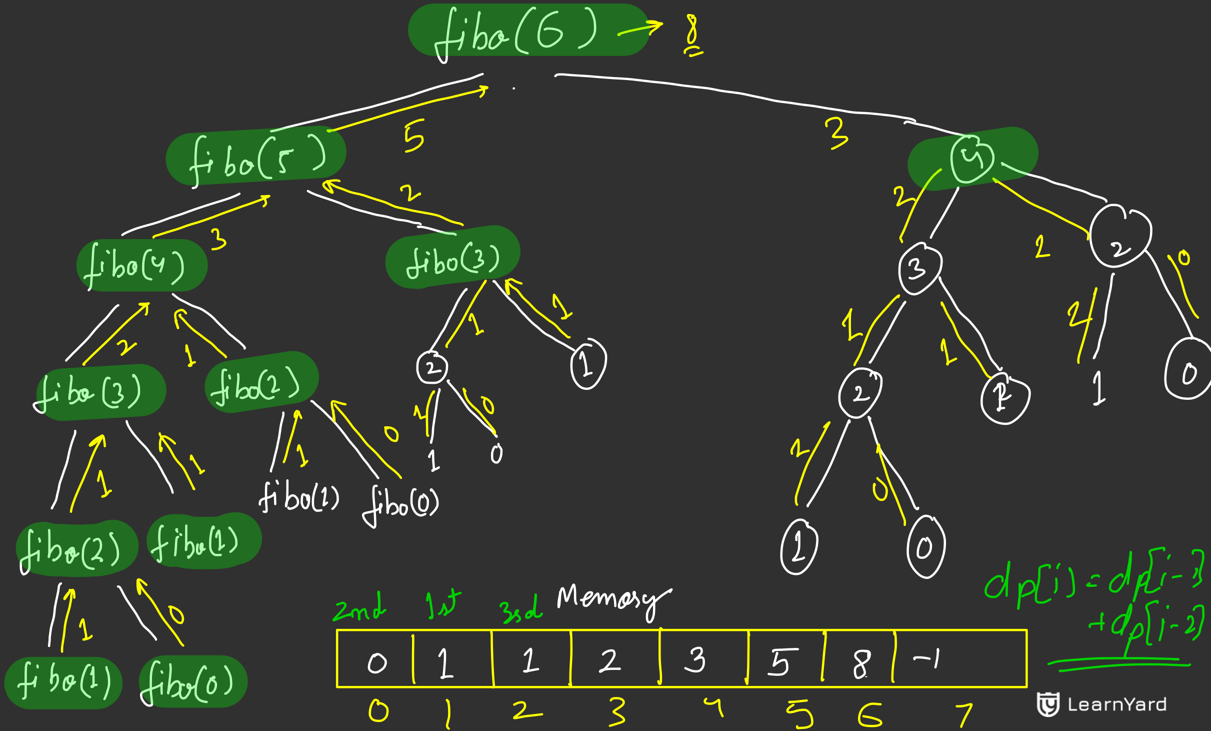
Definition : A technique that combines correctness of complete search & efficiency of greedy.

Let’s understand using an example:
Find Nth fibonacci number

```
int fib(int n) {
    if(n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

Remember the recursion tree and approximate time complexity?

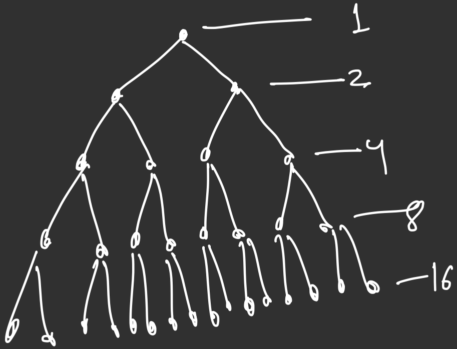
LearnYard



0	1	1	2	3	5	8	-1
0	1	2	3	4	5	6	7

$dp[i] = dp[i-1] + dp[i-2]$

Recursion

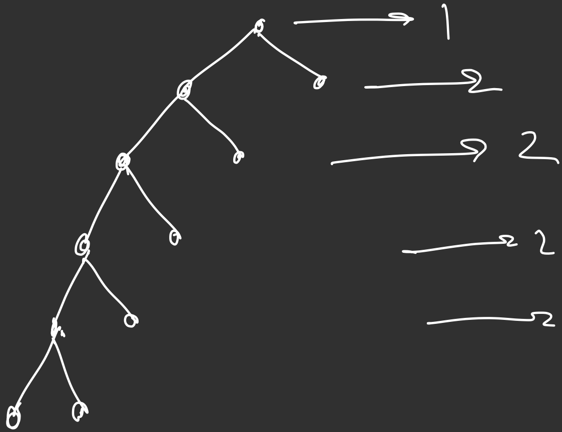


$1 + 2 + 4 + 8 + 16 \cdots$

$2^0 + 2^1 + 2^2 + \cdots 2^n$

$O(\underline{\underline{2^n}})$ $\longrightarrow$

DP



$1 + 2 + 2 + 2 \cdots$

$O(\underline{2n})$

What if we tried to memorise the answer for different problems?

Memoization

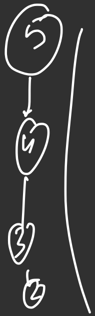
```
ans(n, -1);

int fib(int n) {
    if(ans[n] != -1)          → DP
        return ans[n];

    if(n <= 1)
        return n;   return ans[n] = n;

    ans[n] = fib(n-1) + fib(n-2);
    return ans[n];
}
```

Time Complexity ?

DP is generally helpful only when there are overlapping subproblems

$$\begin{tikzcd}
fact(n) \arrow[d] \\
fact(n-1) \arrow[d] \\
fact(n-2) \arrow[d, dash, dashed] \\
{}
\end{tikzcd}$$

Example, what if we needed to find factorial(n)?



1 last thing before we move to ways to implement DP

Optimal Substructure Property

2 famous ways to implement

Top Down

```
ans(n+1, -1);

int fib(int n) {
    if(ans[n] != -1)
        return ans[n];

    if(n <= 1)
        return n;

    ans[n] = fib(n-1) + fib(n-2);
    return ans[n];
}
```

```
ans[0] = 0;
ans[1] = 1;

for(i = 2; i <= n; i++)
    ans[i] = ans[i-1]
        + ans[i-2];
```

Recursive
with memoization

LearnYard

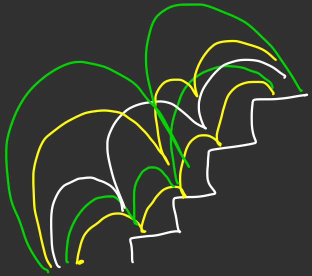
Bottom Up ✓

```
int fib(int n) {
    int ans[n+1]; ✓
    ans[0] = 0, ans[1] = 1;    base case
    for(int i = 2; i <= n; ++i)
        ans[i] = ans[i-1] + ans[i-2];
    return ans[n];
}
```

Iterative

LearnYard

Let’s look at couple of beginner problems to get started



1. Climbing Stairs

LearnYard

Example : N = 4

1st Step -> 1
2nd Step -> 1
3rd Step -> 1
4th Step -> 1

1st Step -> 1
2nd Step -> 1
3rd Step -> 2

1st Step -> 1
2nd Step -> 2
3rd Step -> 1

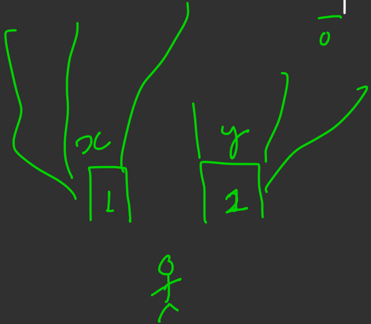
1st Step -> 2
2nd Step -> 1
3rd Step -> 1

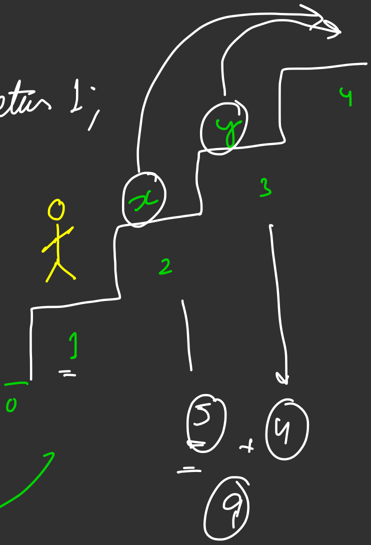
1st Step -> 2
2nd Step -> 2

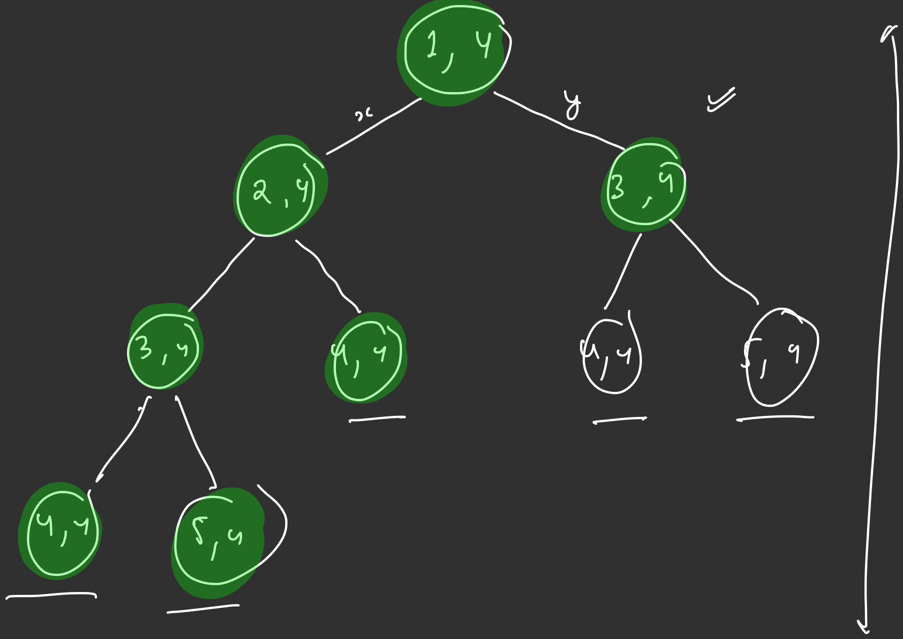
Total Ways = 5

```
func ( int i, int n)
{   if(i > n) return 0; if (i == n) return 1;
    int x = func(i+1, n);
    int y = func(i+2, n);
    ans = x + y;
}
```

Intuition







2^n

$2n$

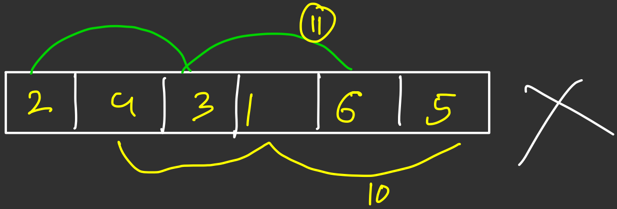
n

How will memoization help?

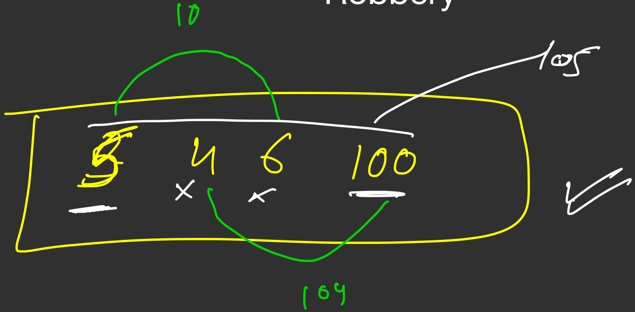
Solution

LearnYard

Let’s implement



2. House Robbery



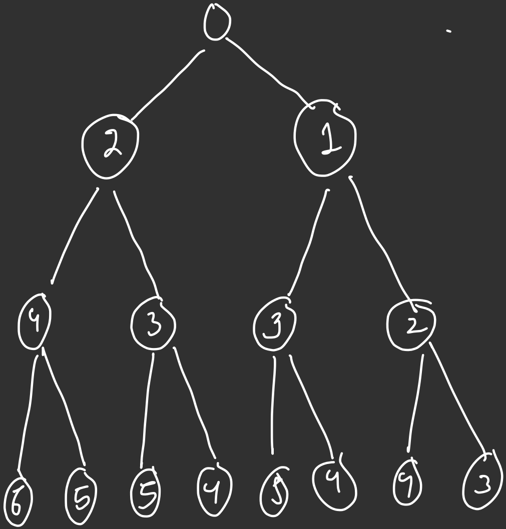
Example

Input:

N = 5
nums = [2,7,9,3,1]

Rob:
House 0 -> 2
House 2 -> 9
House 4 -> 1

ans = 12



Intuition

Solution

Let’s implement

Thank You!

Reminder: Going to the gym & observing the trainer work out can help you know the right technique, but you’ll muscle up only if you lift some weights yourself.

So, PRACTICE, PRACTICE, PRACTICE!

LearnYard