Class 79

Dynamic Programming (DP)

DP = Recursion + optimization (2-3)

Recursion, Backtracking, Tree

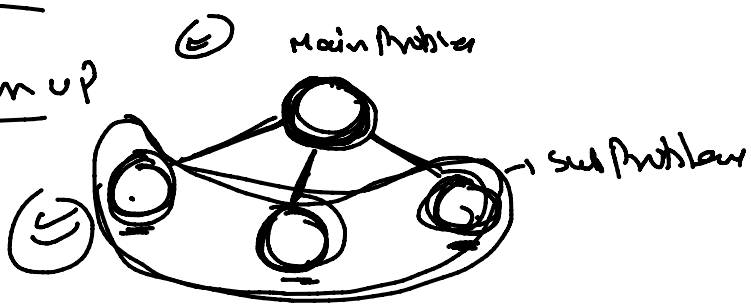
Recursion → Explore all possible ways

T.C ~ Exponential

DP

→ Memoization → Top down

→ Tabulation → Bottom up



DP Problems!

Recursion

Dynamic Programming

→ optimal substructure

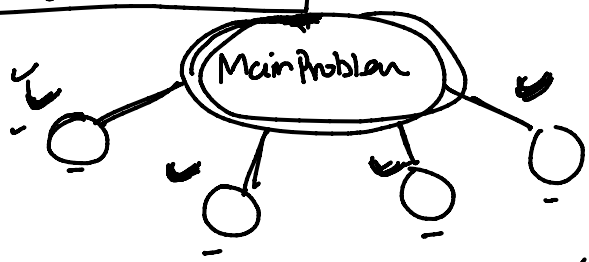
→ overlapping subproblems

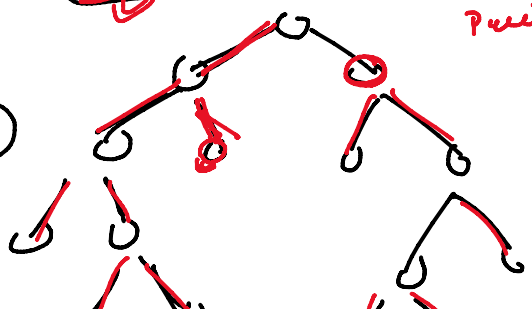
Recursion

Backtracking

one. DP Pruning

No of Pruns.





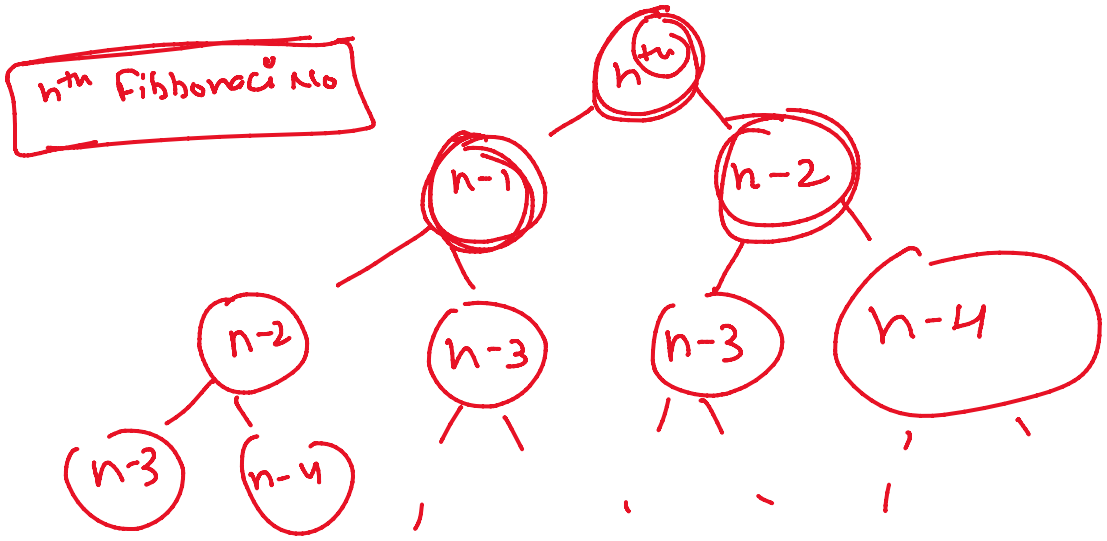
local optimum ↑greedy

local optimum + greedy

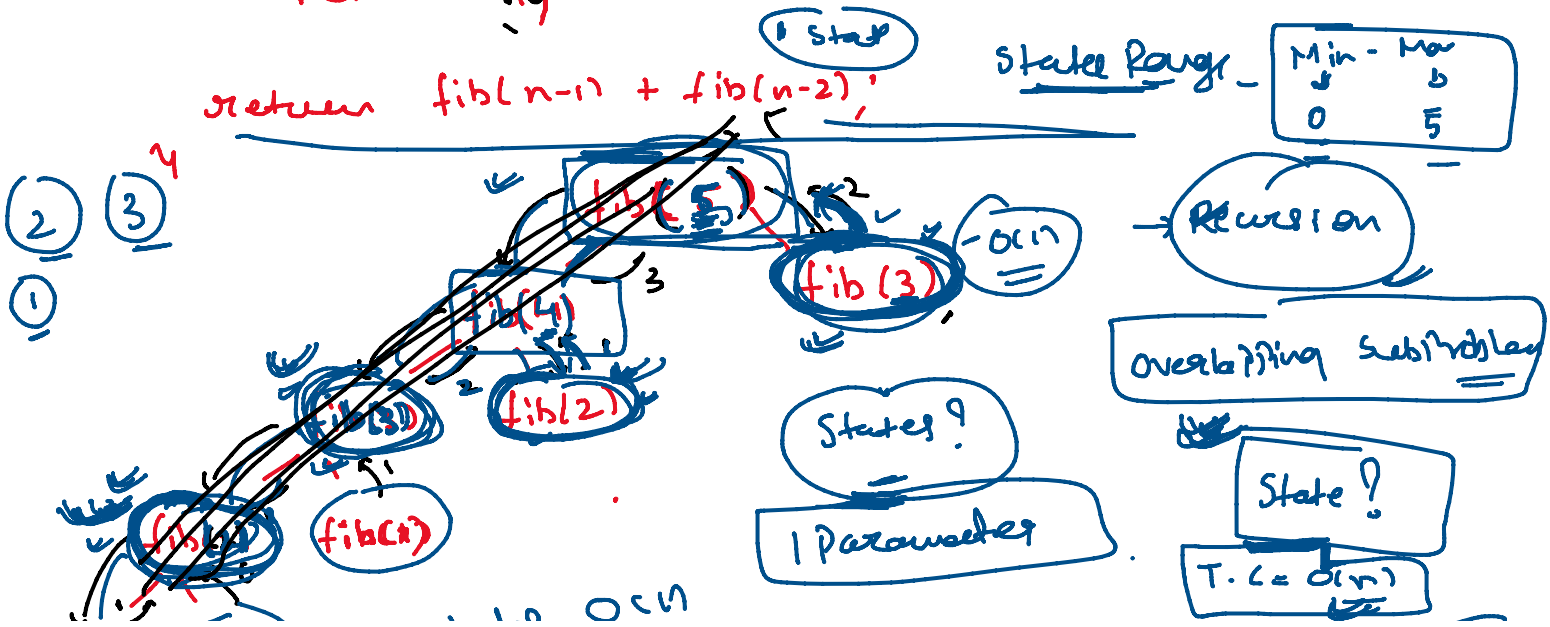
DP → Count total way
→ Optimization
→ Min / Max

Fibbonaci No.

0 1 1 2 3 5 8 13 21 34 - - - .




```
int fib(int n)
{ if(n==0 or n==1)
    return n;
  return fib(n-1) + fib(n-2);
}
```



1 Parameter

T.C = O(n)

n states O(n)

Recursive tree same subproblems again

Reuse subproblems

Dynamic Programming

Every Recursion call has their own stack

S.C = O(n) + O(n)
stack space DP size

Two Recursion calls with same state are same

state 0 - 5

Array/vector of size 6

vector<int> dp(6, -1);

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

dp[0] - dp[2] dp[4]

dp[1] - dp[3] dp[5]

0 - n-1

```
int fib (int n, vector<int>& dp)
{
    if(n==0 or n==1)
        return n
    if(dp[n] != -1) return dp[n];
    dp[n] = fib(n-1) + fib(n-2);
    return dp[n]
}
```

Memoization

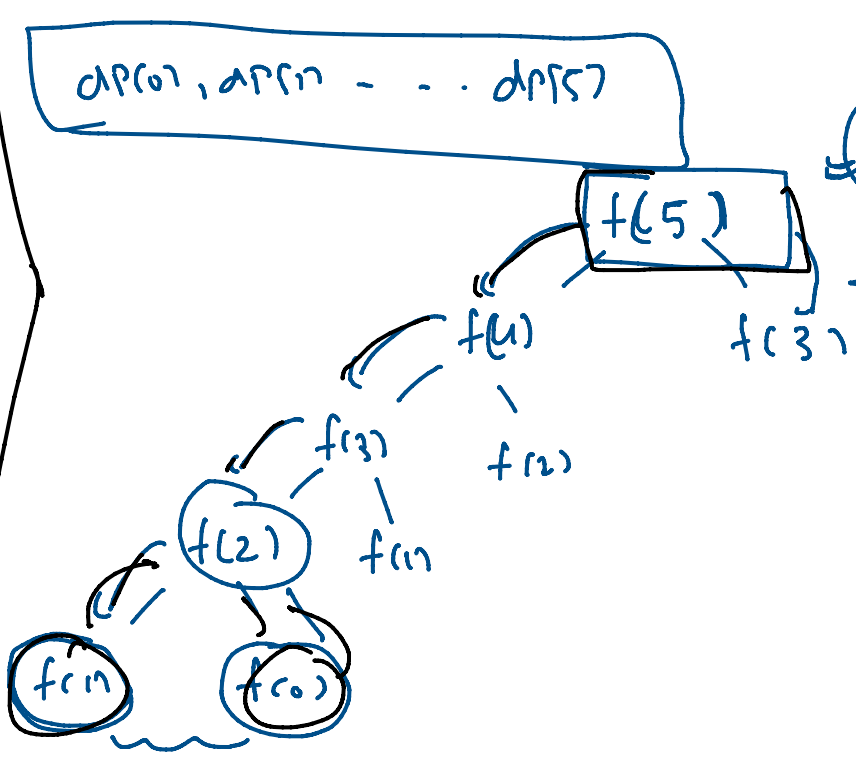
Recursion

Top-down

1D Array

1D inner DP

dp[0], dp[1] - - - dp[5]



1D Array → 1D linear DP

2Dstack → 2D area → 2D DP

3D DP, 4D DP,

Tabulation Approach (Bottom up DP)

We directly start from Bottom;

```
vector<int> dp(n,-1);
dp[0] = 0;
dp[1] = 1;
for (i=2; i<n; i++)
{
    dp[i] = dp[i-1] + dp[i-2];
}
```

T.C = O(n)

S.C = O(n)

Tabulation Approach

optimization

```
int Prev2=0;
int Prev= 1;
```

n = 6

Prev=1, Prev2 = 0

i=2 cur = 1, Prev2 = 1

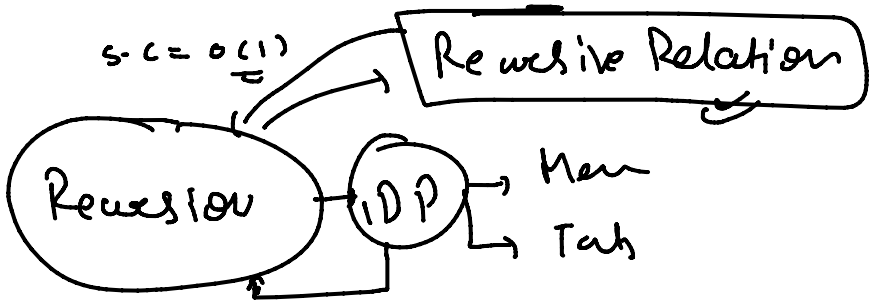
```
int Prev2 = 0;
int Prev = 1;

for (i = 2; i <= n; i++) {
    int cur = Prev + Prev2;
    Prev2 = Prev;
    Prev = cur;
}
return Prev;
```

i = 2 cur = 1, Prev2 = 1, Prev = 1

i = 3 cur = 2, Prev2 = 1, Prev = 2

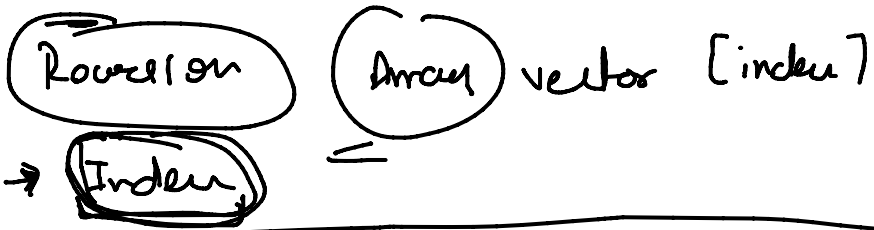
i = 4 cur = 3 Prev2 = 2, Prev = 3

T.C = O(1)



→ Count total ways

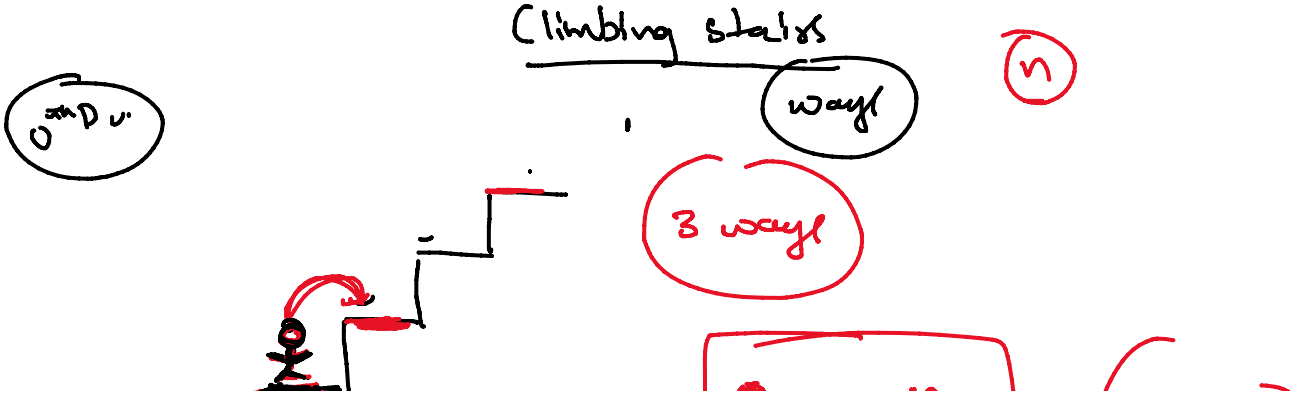
→ Min / Max



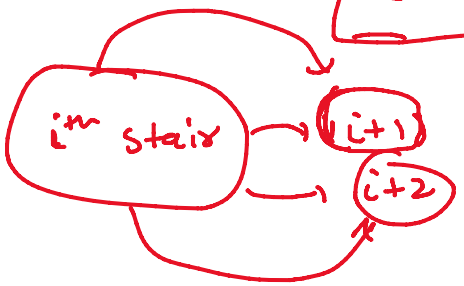
→ Do all possible stuff acc to Problem → Created sub problem

→ Find ans using subproblems

Climbing stairs



indexed



consider each stair as an index

2 DP 3D

int ClimbingStair (int n) ⇒ No of ways to reach n from 0

0 - i^th stair or n from i

1D Array (DP)

```
{
    if (n == 0)
        return 1;
    if (n == 1)
        return 1;
    if (dp[n] != -1) return dp[n];
    dp[n] = ClimbingStair(n-1) + ClimbingStair(n-2);
    return dp[n];
}
```

ways

Climbing Fibbonacci

Tabulation

```
int ClimbingStair(int i, int n)
{
    if (i == n)
        return 1;
    if (i == n-1)
        return 1;
    return ClimbingStair(i+1, n) + ClimbingStair(i+2, n);
}
ClimbingStair(0, n)
```

count total ways

T.C of Memoization == Tabulation