Agenda

1) Expert Mock Interviews
2) Fibonacci Series
3) Introduction To Dynamic Programming
4) Ways To Climb StairCase
5) Minimum No. of squares to get sum = N

Fibonacci Series

0	1	2	3	4	5	6	7	8
0	1	1	2	3	5	8	13	21 ....

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

Base Case ⇒ $fib(0) = 0$

$fib(1) = 1$

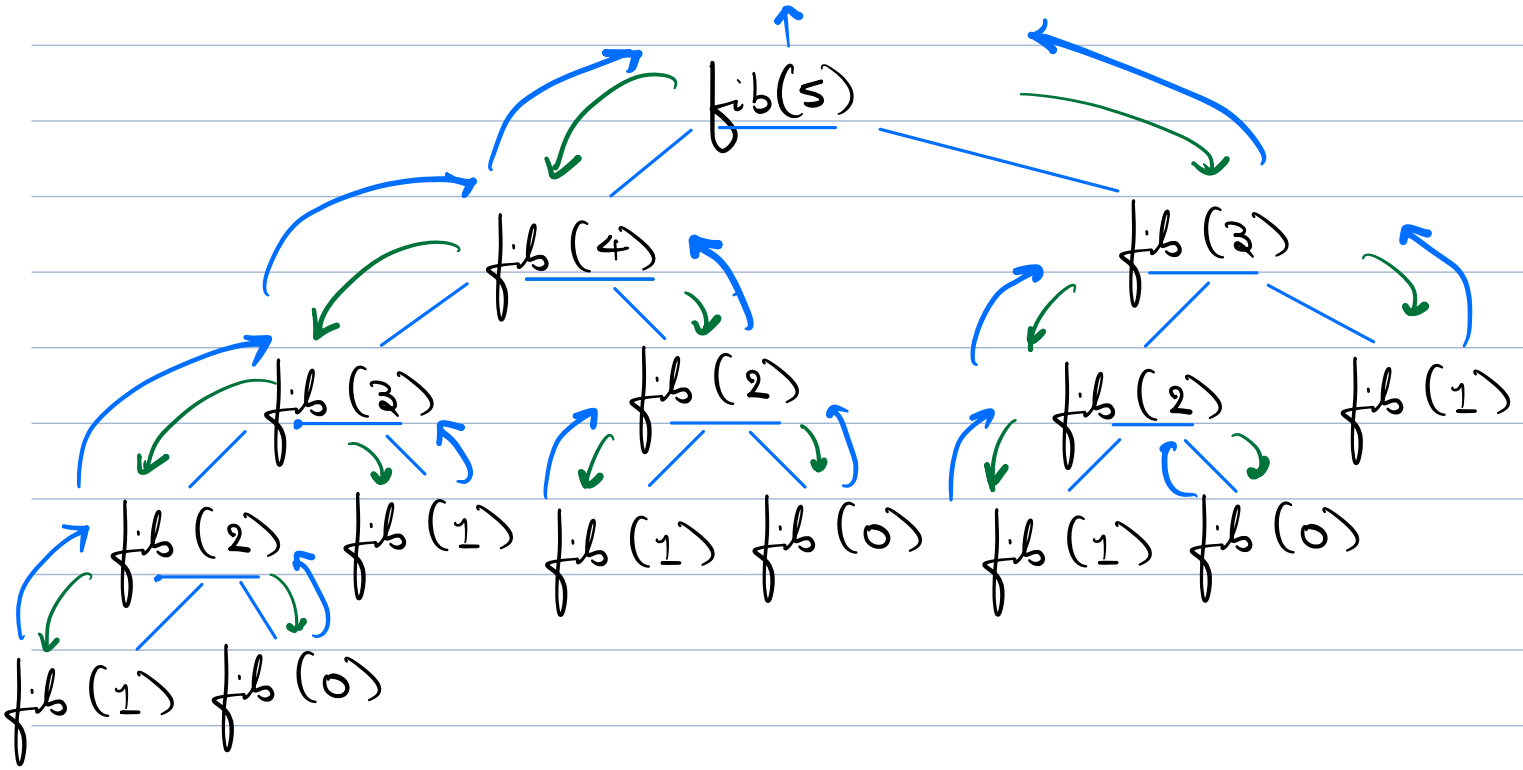
Code

```
int fib(N) {
    if (N==0 || N==1) { return N; }

    return fib(N-1) + fib(N-2);
}
```

T.C. = $O(2^N)$

S.C. = $O(N)$



Anything unoptimal that you are able to see in the above tree ??

Dynamic Programming

⇒ It is an optimisation technique.

⇒ To apply DP, a problem must have 2 properties

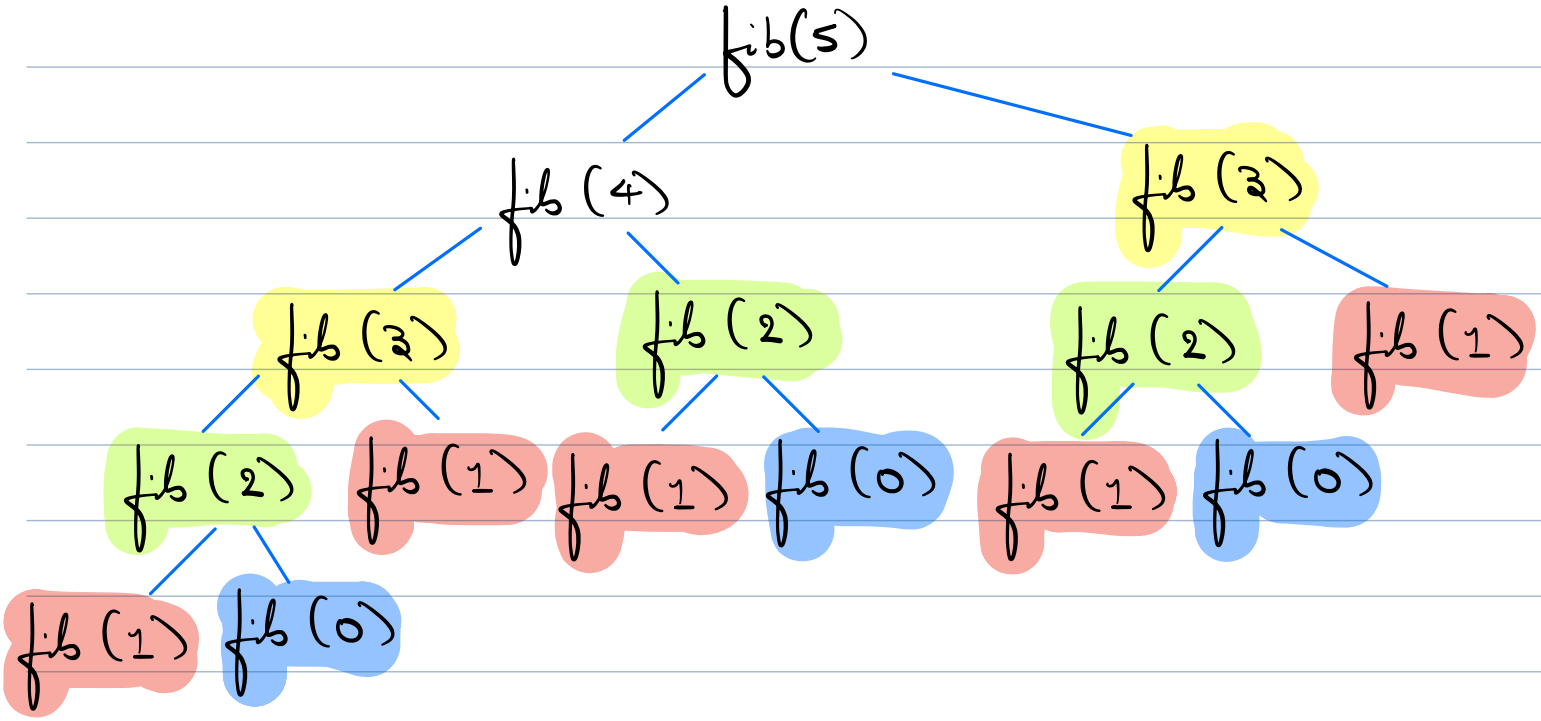
1) Optimal Substructure

⇒ A problem has Optimal Substructure if the solution to the problem can be obtained using the solution of it's subproblem

i.e. Recursion.

2) Overlapping Subproblem

⇒ Calculating similar subproblem multiple times



⇒ If you observe the above two properties, you can apply Dynamic Programming

But How ??

Memoization

⇒ Storing the value of a subproblem that you have computed & use the stored value the next time you encounter the same subproblem again.

for Fibonacci of N

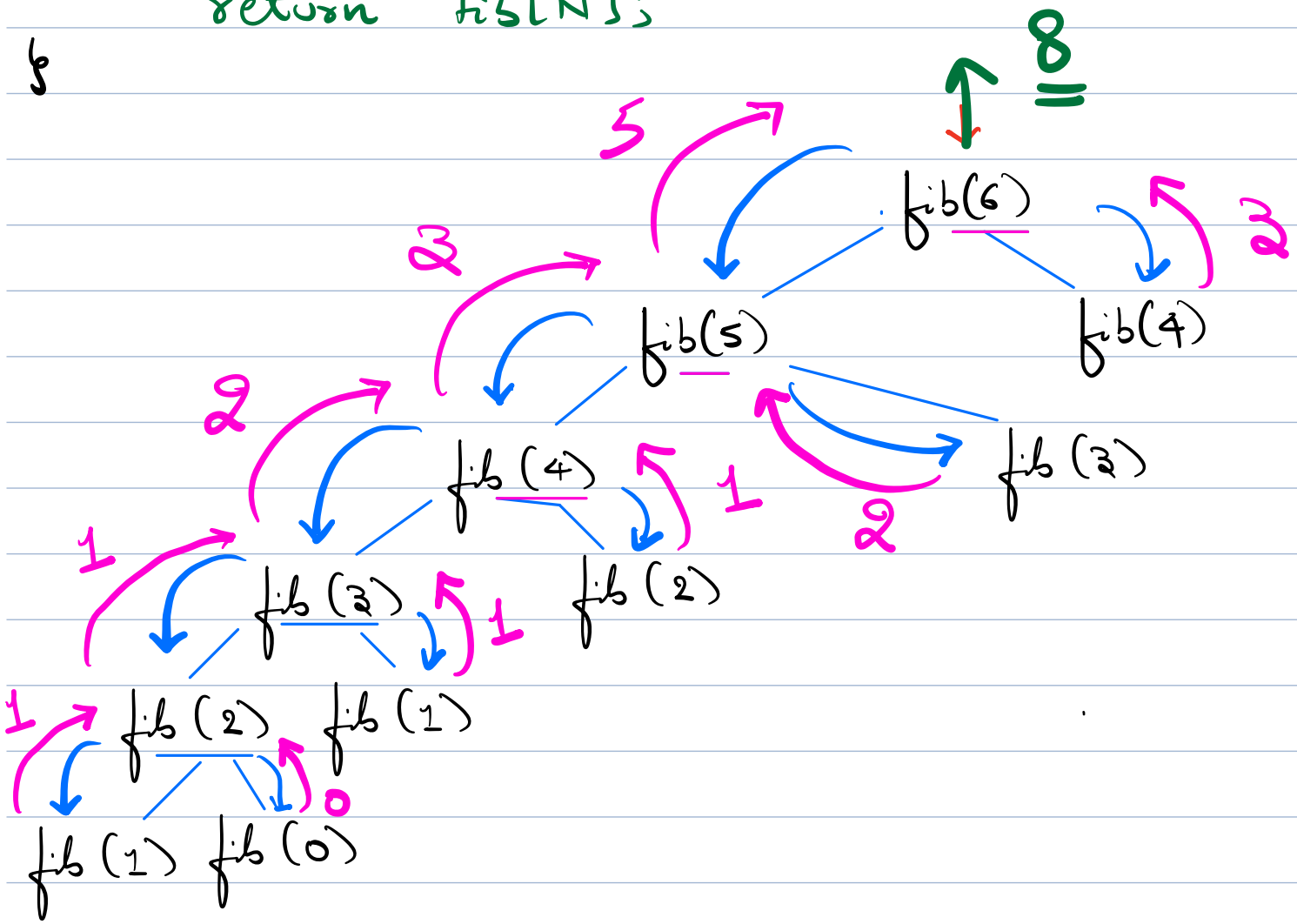
Subproblems ⇒ 0, 1, 2, 3, 4 ..... N

Memoization Array of size N+1

Code

```
fib[N+1] = {-1};
```

```
int fibonacci (N) {
    if (N == 0 || N == 1) { return N; }
    if (Fib[N] != -1) {
        return Fib[N];
    }
    Fib[N] = fibonacci (N-1) + fibonacci (N-2);
    return fib[N];
}
```



Fib[7] = [0 1 2 3 4 5 6
-1 -1, 1, 2, 3, 5, 8,]

T.C. = $O(N)$

S.C. = $O(N) + O(N) = O(N)$

↓ Recursive Stack Space

↓ Memoization

⇒ Recursion + Memoization.

---

2 Ways to Solve DP Problems

1) Top-Down DP

(Recursion + Memoization)

⇒ We start with a bigger problem

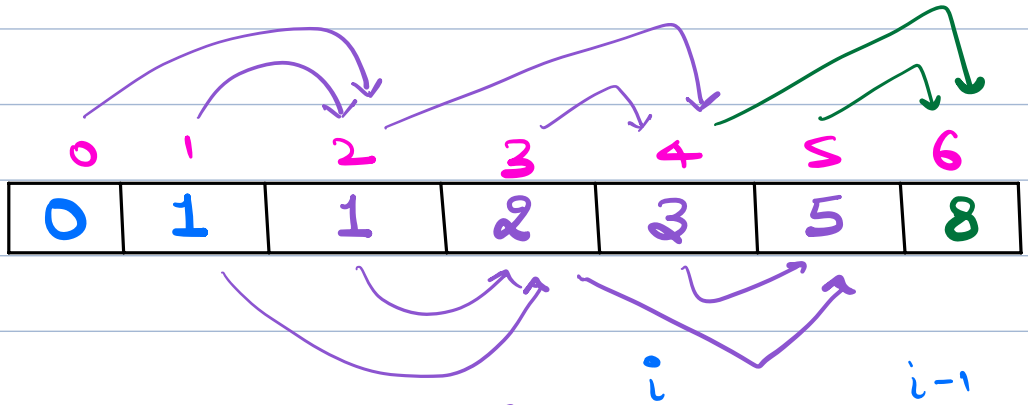
⇒ Go down recursively to smaller subproblems for which we already know the answers (Base Case)

2) Bottom Up DP

⇒ Iterative DP. (Tabulation)

⇒ Start from the smallest subproblem for which we already know the answer (Base Case) & use that to iteratively solve the bigger problems.

fib(6)



Recursive relation ⇒ $fib(N) = fib(N-1) + fib(N-2)$

Code

```
fib[N+1];  ←
fib[0] = 0;
fib[1] = 1;
for (i=2;  i<=N;  i++) {
        fib[i] = fib[i-1] + fib[i-2];
}

return  fib[N];
```

T.C. = O(N)

S.C. = O(N)

↓

Tabulation.

Note: In Bottom up DP, we need to be aware about the order in which the states needs to be computed.

Bottom Up v/s Top Down

Bottom Up	Top Down
1) Less Intuitive	1) More Intuitive ↑
2) More Space efficient	2) Less Space Efficient
3) More Control	3) Less Control

More Control = More Optimisation

Q) Can we have a more space efficient solution for fibonacci ??

Code

```
int fibn1 = 1;
int fibn2 = 0;

for (i = 2; i <= N; i++) {
    int temp = fibn1 + fibn2;
    fibn2 = fibn1;
    fibn1 = temp;
}
return fibn1
```

T.C. = O(N)
S.C. = O(1)

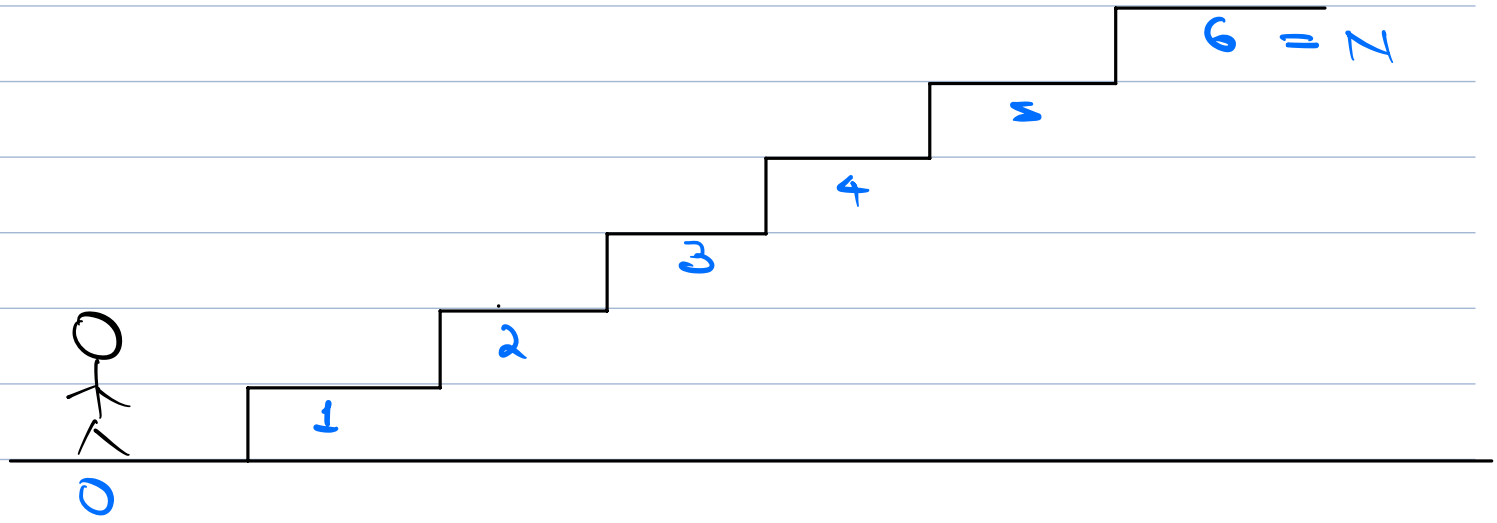
DP State

DP[i] ⇒ What are you computing in ith step.

DP[i] = ith fibonacci number = fib[i]
= fib[i-1] + fib[i-2]

DP[i] = DP[i-1] + DP[i-2]

Staircase



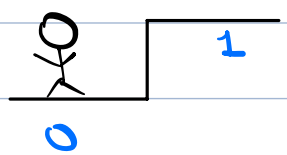
One step ⇒ Jump 1 stair case or Jump 2 stair cases.

find the no. of ways to reach Nth staircase.

Eg

Eg

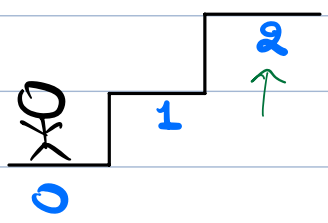
N = 1



Ans = 1

0 → 1

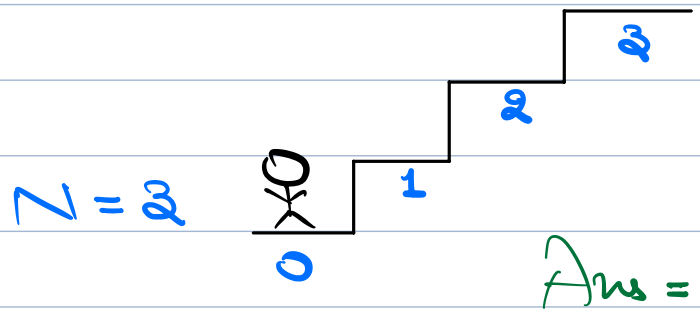
N = 2



0 → 1 → 2

0 → 2

Ans = 2



$0 \rightarrow 1 \rightarrow 2 \rightarrow 3$

$0 \rightarrow 1 \rightarrow 3$

$0 \rightarrow 2 \rightarrow 3$

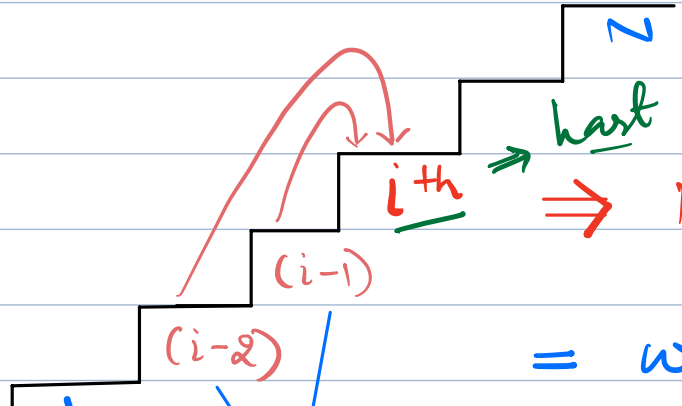
Ans = 3

Sol^n Notion of Choice $\Rightarrow$ Recursive Solution.

★ Steps to Solve ~~DP~~ Recursive problem

1) Elements of Choice $\Rightarrow$ Step of size 1
OR
Step of size 2

2) How to represent a state
(What my state represents)



$i^{th} \Rightarrow$ No. of ways to reach i^{th} staircase.

$= ways(i)$

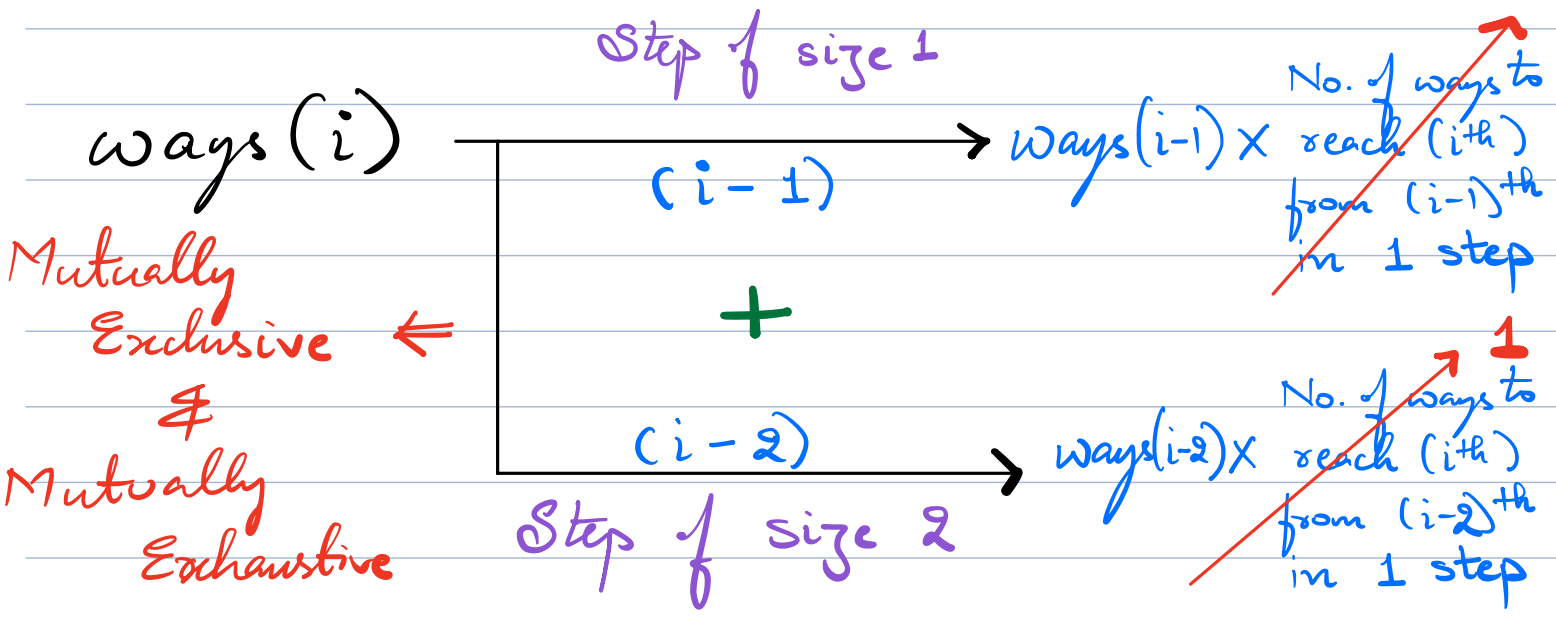
penultimate states (2nd last states)

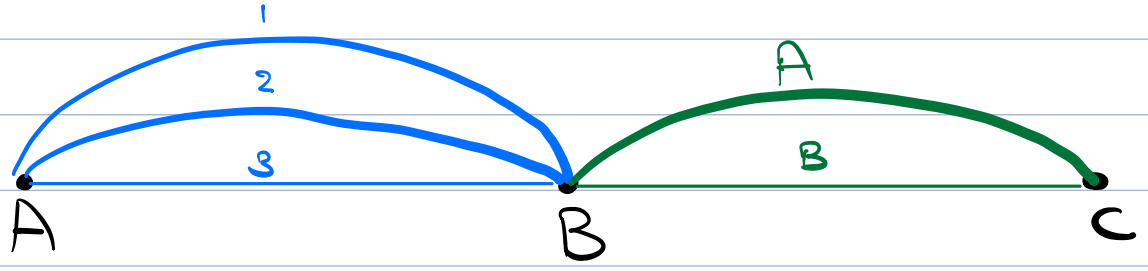
3) Use step 1 & step 2 to write recurrance relationship.

4) Which state is the final answer of the problem ??

= ways (N)

Step 3 ⇒ Recurrance rel^n





unique ways to go from A to C

1A	2A	3A
1B	2B	3B

= ways (A-B) × ways (B-C)

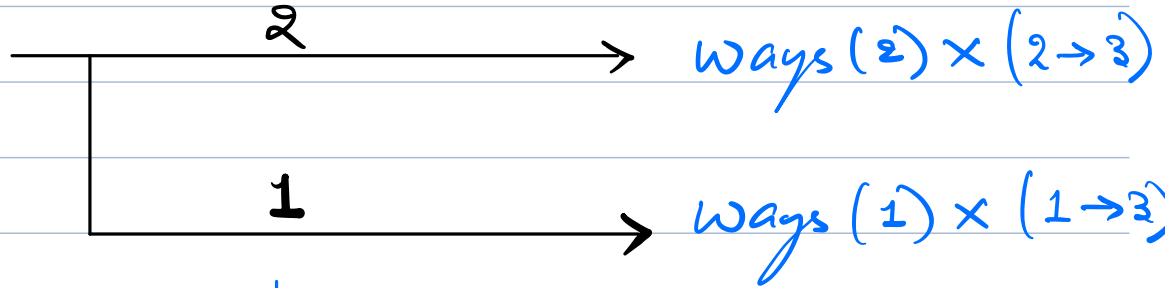
$$Ways(i) = Ways(i-1) + Ways(i-2)$$

↳ Fibonacci relationship.

1) Mutually Exclusive

⇒ Choices should not overlap.

N = 3



2 ⇒ 0 → 1 → 2 → 3

0 → 2 → 3

1 ⇒ 0 → 1 → 3 (1)

0 → 1 → 2 → 3 Twice

2) Mutually Exhaustive

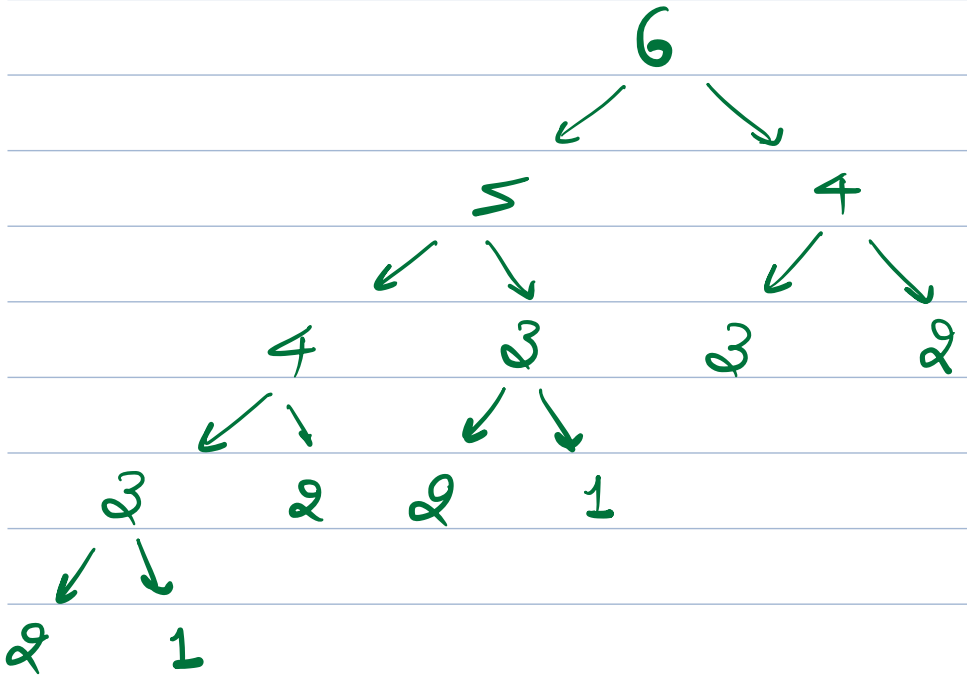
⇒ Choices should cover all the cases.

Base Case .

$N = 1$ (Ans = 1)

$N = 2$ (Ans = 2)

Is it possible to apply DP ??



⇒ Overlapping Subproblems

⇓

We can apply DP

```
DP[N+1] = {-1};

int ways(N) {
    if (N == 1 || N == 2) { return N; }
    if (DP[N] != -1) {
        return DP[N];
    }
    DP[N] = ways(N-1) + ways(N-2);
    return DP[N];
}
```

H.W: Iterative Sol^n

Given an integer N.
Figure out the minimum no. of perfect squares we can add to get sum = N.

$N = 6 \Rightarrow 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 \Rightarrow 6$

$2^2 + 1^2 + 1^2 \Rightarrow 3$ Ans

$N = 10 \Rightarrow 1^2 + 1^2 \ldots$ (10 times) $\Rightarrow 10$

$2^2 + (1^2 + 1^2 \ldots$ 6 times$) \Rightarrow 7$

$2^2 + 2^2 + 1^2 + 1^2 \Rightarrow 4$

$3^2 + 1^2 \Rightarrow 2$ Ans

$N = 9 \Rightarrow 1^2 + 1^2 \ldots$ 9 times $\Rightarrow 9$

$2^2 + (1^2 \ldots$ 5 times$) \Rightarrow 6$

$2^2 + 2^2 + 1^2 \Rightarrow 3$

$3^2 \Rightarrow 1$ Ans

Sol^n

$N -$ (Nearest Perfect Square)

$\Downarrow$

Square of largest no. whose square is $\leq N$.

$N = 6 - \underline{2^2} = 2 - \underline{1^2} = 1 - \underline{1^2} = \underline{0}$

Ans = 3

Not correct

$N = 12 - \underline{3^2} = 3 - \underline{1^2} = 2 - \underline{1^2} = 1 - \underline{1^2} = 0$

4 ?? Ans

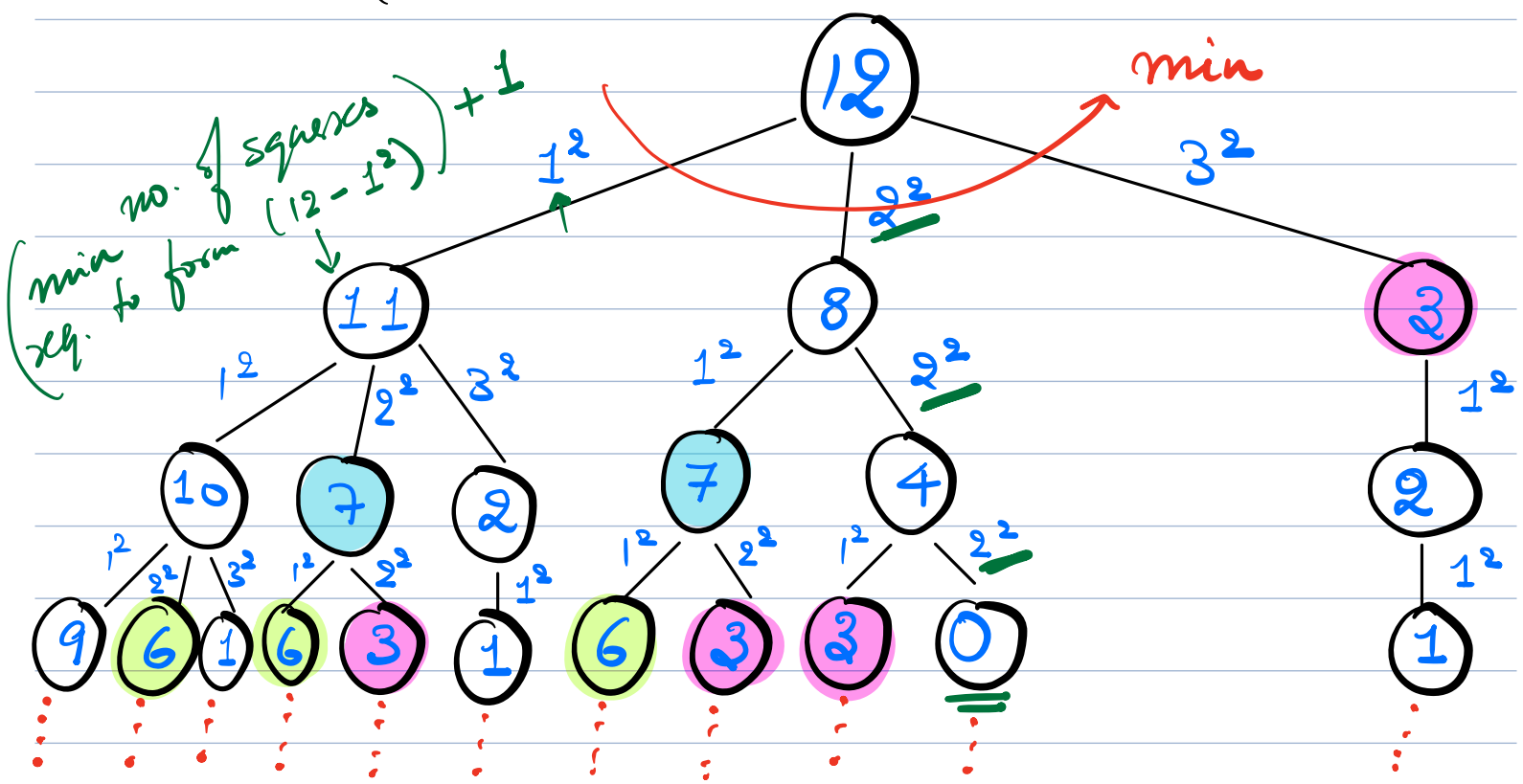
$12 \Rightarrow 2^2 + 2^2 + 2^2 \Rightarrow 3$ Ans

What to do when Greedy fails.??

$\Rightarrow$ Brute force (All Choices)

Recursion

$N - (\forall i$ whose square $\leq N)$



1) Elements of Choice (N) ⇒ ∀i where i×i ≤ N

2) What will the state represent ⇒ Min squares required to form sum = N

minSquare(i) ⇒ Min squares required to form sum = i

3) Recurrance relationship.

$$minSquare(i) = \forall_{j \Rightarrow j \times j \leq i} \; min(minSquare(i - j^2) + 1)$$

4) Which state is the answer ??

minSquare(N)

N ⇒ 0, 1, 2, 3 ..... N

DP Array of size N+1

Code

```
DP[N+1] = {-1};

int minSquare(N) {
    if (N==0) { return 0; }

    if (DP[N] != -1) {
        return DP[N];
    }

    ans = INT_MAX;

    for (i=1; i*i<=N; i++) {
        ans = min(ans, minSquare(N-i^2)+1);
    }

    DP[N] = ans;
```

return ans;

}

T.C. = O(N) x O(sqrt(N))

S.C. = O(N)