Dynamic Programming 1

TABLE OF CONTENTS

Notes

DSA - 4 completion

↓

Full Syllabus Contest → 5 problems

[4/5 problems → clearance]

↓

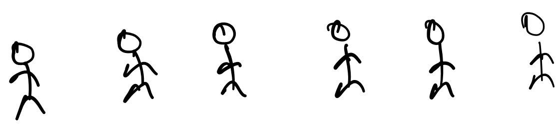
A.I Mock Interview

↓

DSA certified.

Dynamic Programming

5 4 9 7 6 → 31



5 4 9 7 6 8 → 39

⇒ use pre-calculated value & don't calculate it again.

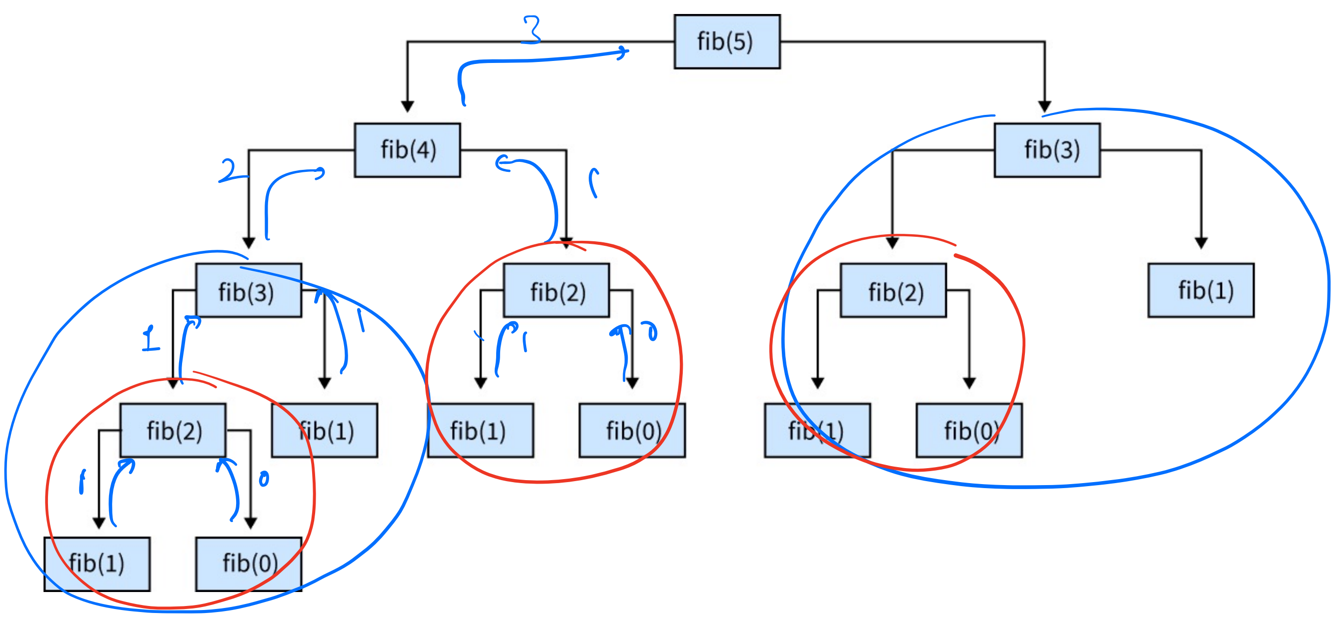
Nth Fibonacci Number

0	1	1	2	3	5	8	13	21	34	55
0	1	2	3	4	5	6	7	8	9	10

< / > Code

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

T.C → $O(2^N)$
S.C → $O(N)$



idea → We need to store result of already solved problems and use it.

Dynamic Programming → optimisation over recursion.

1. Optimal Structure → solving a problem using smaller instances of the same problem.

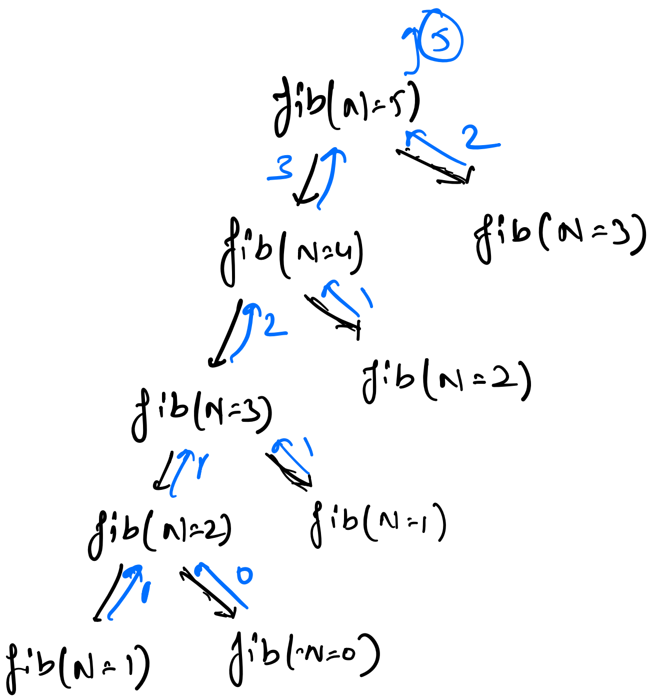
2. Overlapping Subproblems → solving same sub-problems again & again.

Code:

```
int dp[N+1],      ∀i, dp[i] = -1;
int fib( int n , int[] dp){
    if(n==0 || n==1){ return n}
    if(dp[n] != -1){ return dp[n] }
    dp[n] = fib(n-1, dp) + fib(n-2, dp);
    return dp[n];
}
```

dp →

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



T.C → O(N)
S.C → O(N)

Top - down Approach

↳ starting from largest / biggest problem

↳ Recursion

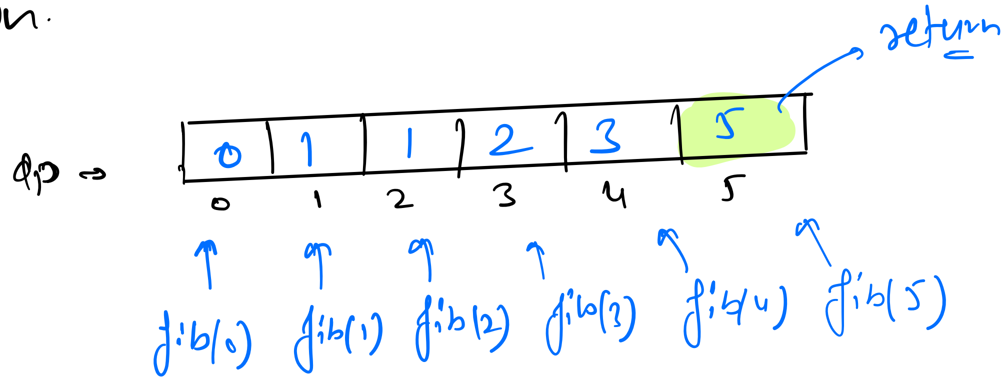
↳ Memoization

Bottom - up Approach

↳ starting from smallest problem.

↳ iterative

↳ Tabulation.



Code for Bottom-Up Approach

```
dp [N+1];
dp [0] = 0,  dp[1] = 1;
for(i = 2;  i ≤ N;  i++){
        dp[i] = dp[i-1] + dp[i-2];
}
return dp[N];
```

T.C → O(N)
S.C → O(N)

Further S.C Optimisation?

0 1 1 2 3 5

a b

Code:-

```
a = 0
b = 1

for( i = 2; i <= N; i++){
    int c = a+b;
    a = b;
    b = c;
}

return b;
```

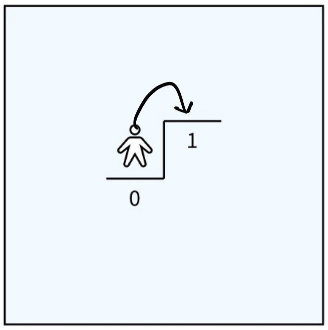
T.C → O(N)
S.C → O(1)

Climbing Stairs

$1 \leq N \leq 10^5$

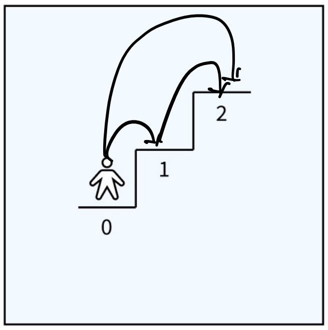
Calculate the number of ways to reach Nth stair. You can take 1 step - at a time or 2 steps at a time.

N = 1



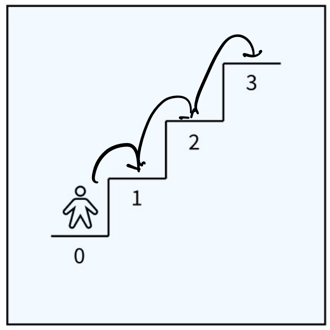
ans = 1

N = 2



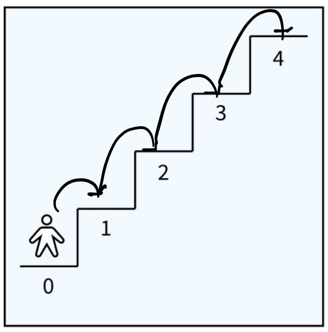
ans = 2

N = 3



111
12
21

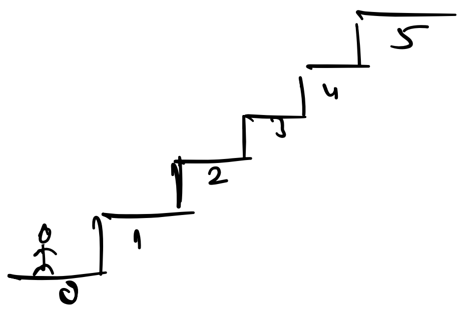
N = 4



ans = 5

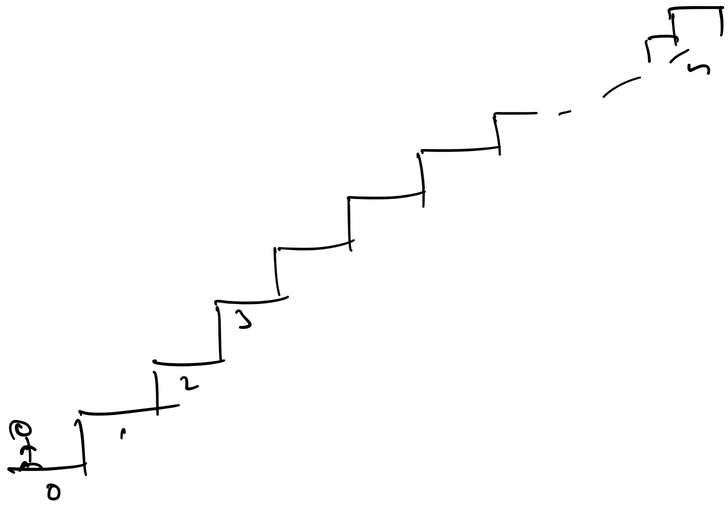
1111
112
121
211
22

N = 5



1111
1112
1121
1211
122

2111
212
221



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

#code →

```
int ways(int N){
    if(N==1 || N==2){return N}
    return ways(N-1) + ways(N-2)
}
```

↓

optimization (todo)

As it is similar to fib

< Question > : Find the minimum number of perfect squares required to get sum = N?

[numbers can repeat]

N = 6

$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 \rightarrow 6$

$2^2 + 1^2 + 1^2 \rightarrow 3$

ans = 3

N = 10

$1^2 + 1^2 + 1^2 + \_\_ \_\_ - 1^2 \rightarrow 10$

$2^2 + 1^2 + 1^2 + \_ \_ 1^2 \rightarrow 7$

$2^2 + 2^2 + 1^2 + 1^2 \rightarrow 4$

$3^2 + 1^2 \rightarrow 2$

ans = 2.

N = 9

$3^2 \rightarrow$ ans = 1

N = 5

$1^2 + 1^2 + 1^2 + 1^2 + 1^2 \rightarrow 5$

$2^2 + 1^2 \rightarrow 2$

ans = 2

Idea → $\boxed{\text{N - nearest perfect square.}}$ → Greedy ✗

6 $\xrightarrow{-2^2}$ 2 $\xrightarrow{-1}$ 1 $\xrightarrow{-1}$ 0

10 $\xrightarrow{-9}$ 1 $\xrightarrow{-1}$ 0

5 $\xrightarrow{-4}$ 1 $\xrightarrow{-1}$ 0

12 $\xrightarrow{-9}$ 3 $\xrightarrow{-1}$ 2 $\xrightarrow{-1}$ 1 $\xrightarrow{-1}$ 0 } 4

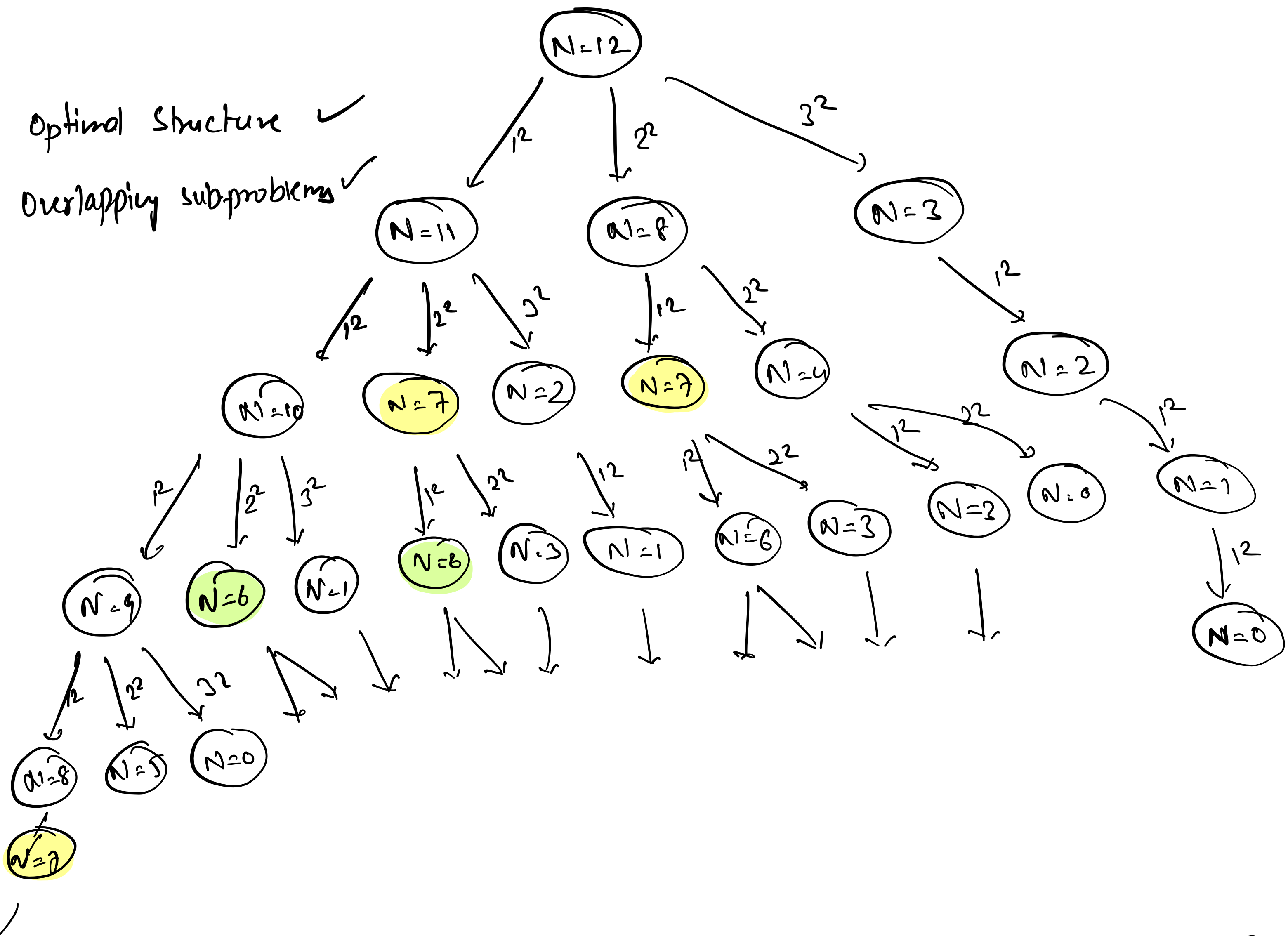
$12 \rightarrow 2^2 + 2^2 + 2^2$

But answer is 3.

Idea -1 Try every possible way to form the answer

Optimal Structure ✓

Overlapping subproblems ✓



$$minPsq(12) = min\left[minPsq(12-1^2),\ minPsq(12-2^2),\ minPsq(12-3^2)\right] + 1$$

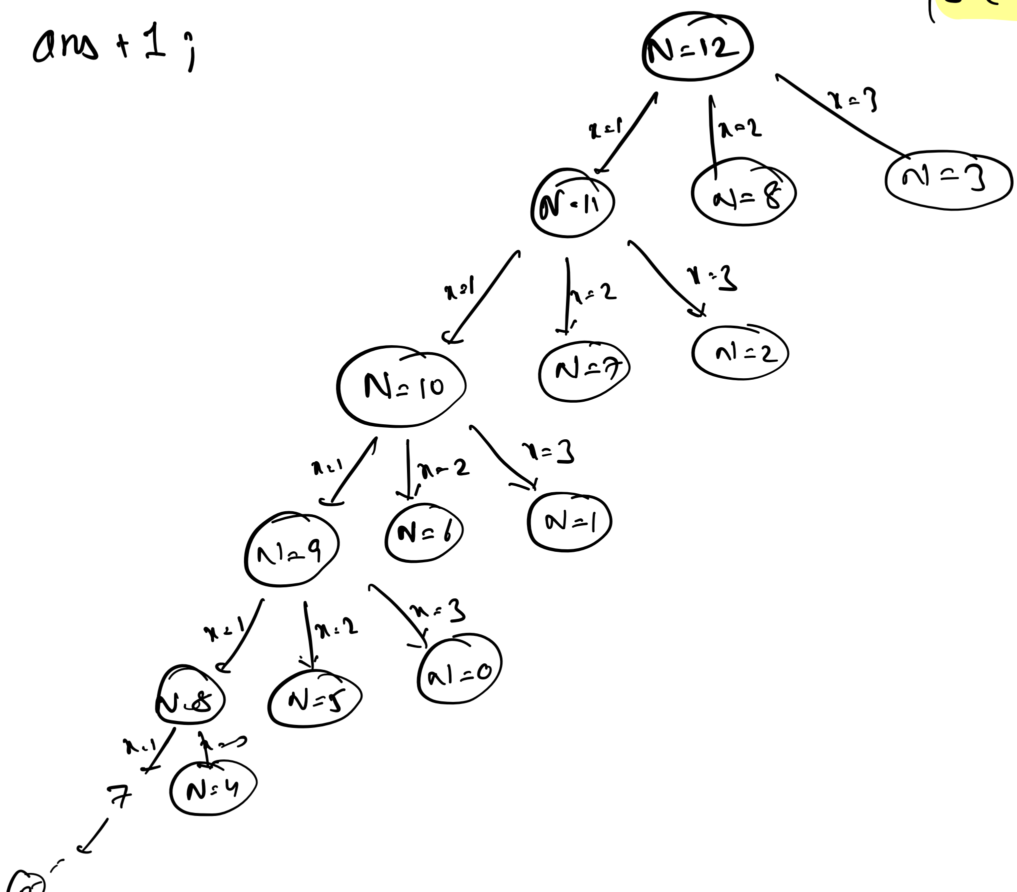
$$\boxed{minPsq(N) = \min_{x^2 \le N}\left[minPsq(N - x^2)\right] + 1}$$

SCALER

</> Code

```
int dp[N+1].     ∀i, dp[i] = -1;

int minPSq( int N, int[] dp) {
    if (N == 0) { return 0 }

    if( dp[N] != -1) { return dp[N] }
    ans = ∞

    for( x = 1; x * x <= N ; x++) {
        ans = min(ans, minPSq( N - x*x, dp ));
    }

    dp[N] = (ans + 1);
    return ans + 1;
}
```

T.C → O(N √N)
S.C → O(N)



Bottom - up

return

0	1	2	3	1	2	3	4	2	1	2	3	3	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

-1^2 -2^2

dp[i] - Min perfect square required to form i

< / > Code

```
dp[N+1];

dp[0] = 0;

for( i = 1; i <= N; i++){
    ans = ∞
    for( x = 1; x*x <= i ; x++){
            ans = Min( ans, dp[i - x*x]);
    }
    dp[i] = ans + 1;
}

return dp[N];
```

T.C → $O(N \cdot \sqrt{N})$
S.C → $O(N)$

3 magical steps →

1. decide storage
2. store ans in storage before returning it.
3. Before making the recursive call, check if the answer is pre-calculated.