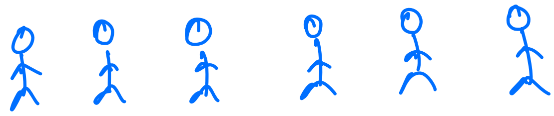
Dynamic Programming

5 4 9 7 6 = 31



5 4 9 7 6 10 = 41

$$psum[i] = psum[i-1] + arr[i]$$

→ use pre-calculated value and don't calculate it again.

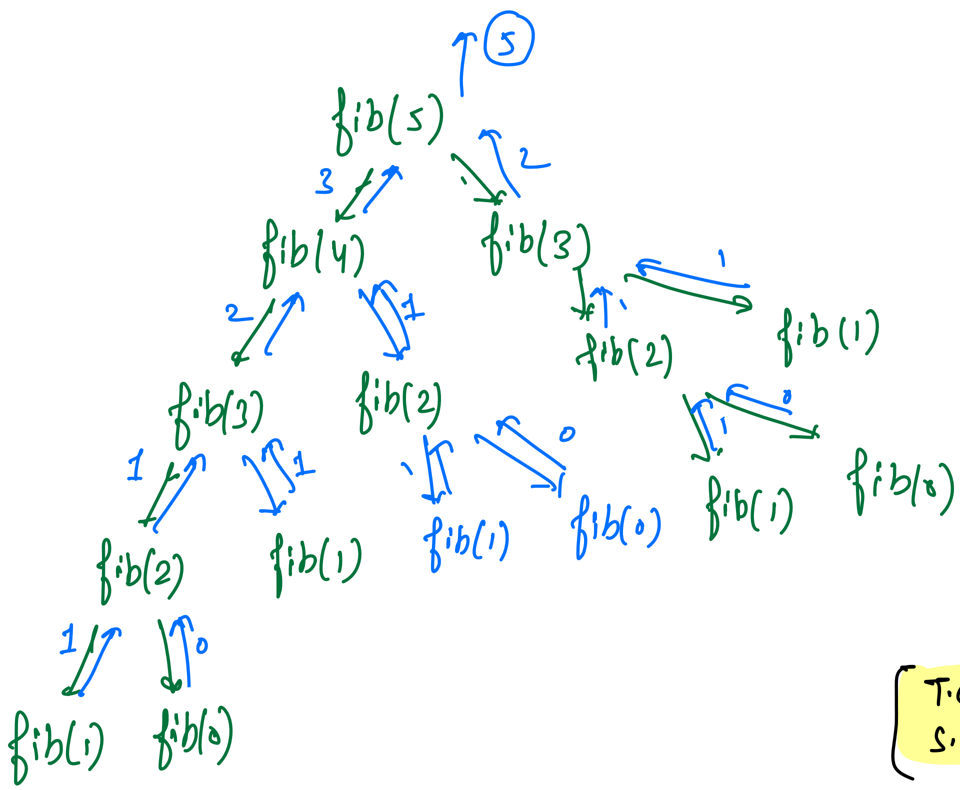
N^{th} Fibonacci Number

N=10

0 1 1 2 3 5 8 13 21 34 55

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

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



T.C → $O(2^N)$
S.C → $O(N)$

① optimal substructure → solving a solving using smaller instances of same problem.

② overlapping sub-problems → solving a sub-problem again & again.

Idea → store the result of already solved problems & use it.

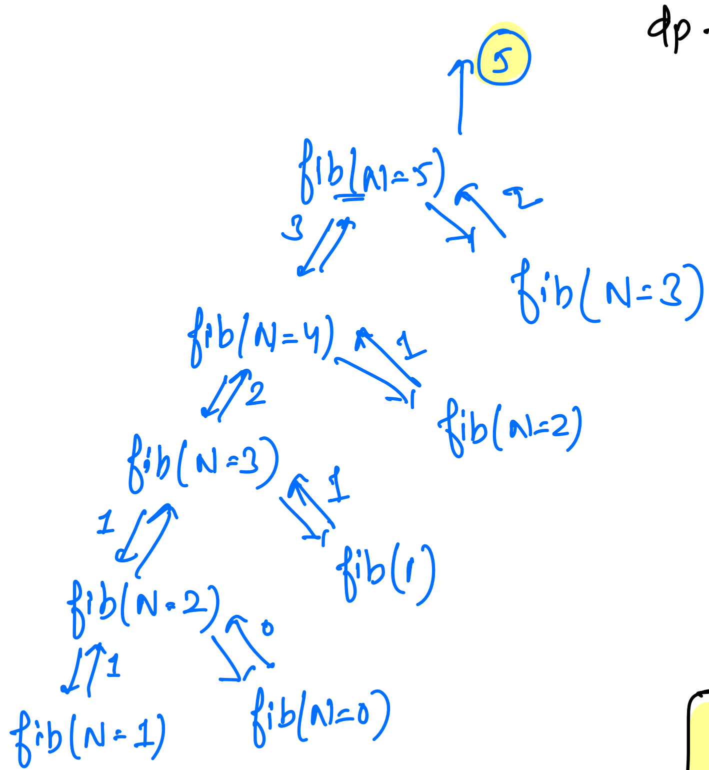
```
int dp[N+1];  // ∀i, dp[i] = -1

int fib( int N , int[] dp){
    if(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



$$\boxed{\begin{array}{l} T.C \rightarrow O(N) \\ S.C \rightarrow O(N) \end{array}}$$

D.P. → optimisation over recursion

Top-down Approach

↳ starting from largest/biggest problem.

- ↳ Recursion
- ↳ Memoization

Bottom-Up Approach

↳ starting from smallest problem.

- ↳ iterative
- ↳ Tabulation

ans.

dp →

0	1	1	2	3	5
0	1	2	3	4	5
↑	↑	↑	↑	↑	↑
fib(0)	fib(1)	fib(2)	fib(3)	fib(4)	fib(5)

code →

```
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 ? → Yes.

N=5.

a=0, b=1;

0 1 1 2 3 5

↑ a ↑ b

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

$$\boxed{\begin{aligned} T.C &\to O(N) \\ S.C &\to O(1) \end{aligned}}$$

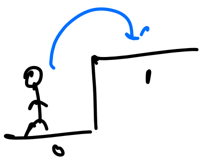
Climbing Stairs

$1 \le N \le 10^5$

N stairs → 1 jump, 2 jump

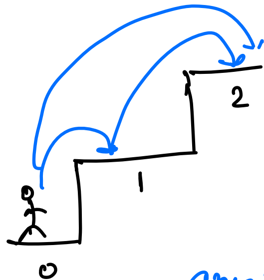
No. of ways to reach N^{th} stair?

$N=1.$



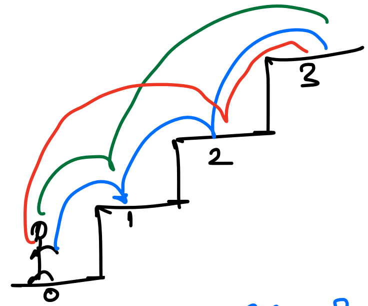
ans = 1.

$N=2.$



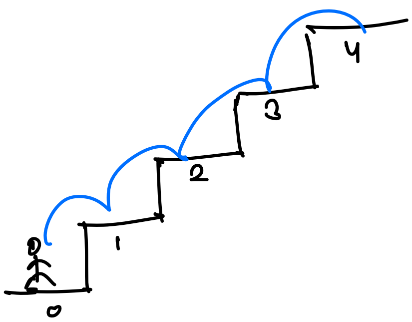
ans = 2.

$N=3.$



ans = 3.

$N=4.$



ans = 5.

1 1 1 1
1 1 2
1 2 1
2 1 1
2 2

idea → Try out all the possibilities / possible ways.

$N \xrightarrow{\text{jump}=1} N-1$

$N \xrightarrow{\text{jump}=2} N-2$

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

ways to climb N stairs =

take a jump of 1 and find ways to climb N-1 stairs

or

take a jump of 2 and find ways to climb N-2 stairs.

$ways(N) = ways(N-1) + ways(N-2)$] similar to fibonacci

$N < 0$ → return 0

$N = 0$ → return 1

Q: Find the minimum no. of perfect squares required to get sum = N? [no.'s can repeat]

N = 6.

$1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 = 6.$

$1^2 + 1^2 + 2^2 = 6$

ans = 3.

N = 10

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

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

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

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

ans = 2.

N = 9

3^2

ans = 1.

N − nearest perfect square [Greedy] ✗

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

$9 \xrightarrow{-9} 0$

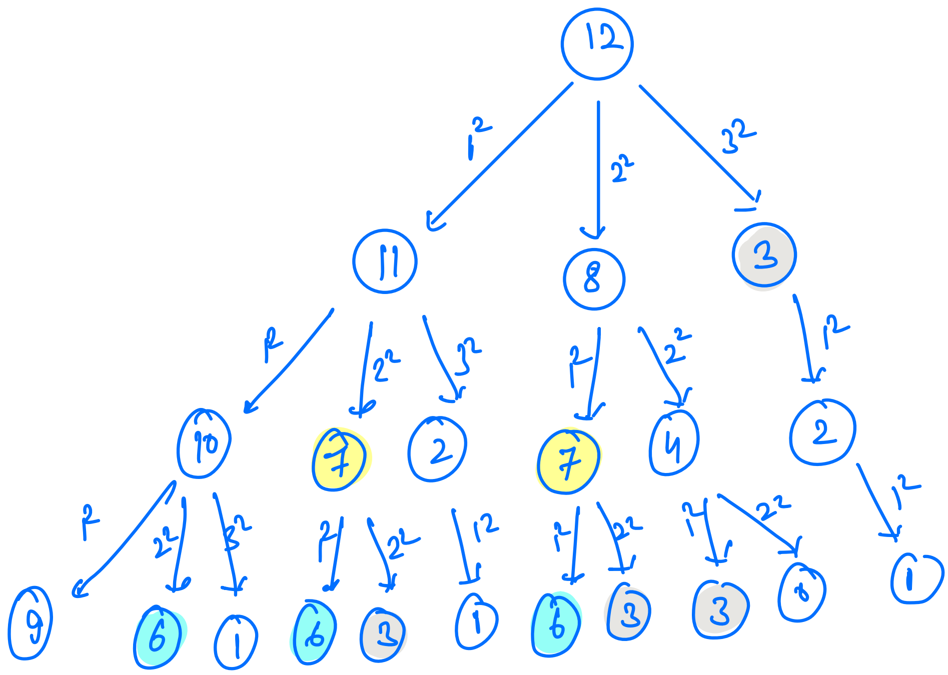
$12 \xrightarrow{-9} 3 \xrightarrow{-1} 2 \xrightarrow{-1} 1 \xrightarrow{-1} 0$

N = 12.

$2^2 + 2^2 + 2^2 = 12.$

ans = 3.

idea. → try every possible way to form the ans.



optimal sub-structure ✓

+

overlapping sub-problems ✓

$$minPSq(12) = Min\left[minPSq(11),\ minPSq(8),\ minPSq(3)\right] + 1$$

$11 = 12-1^2$, $8 = 12-2^2$, $3 = 12-3^2$

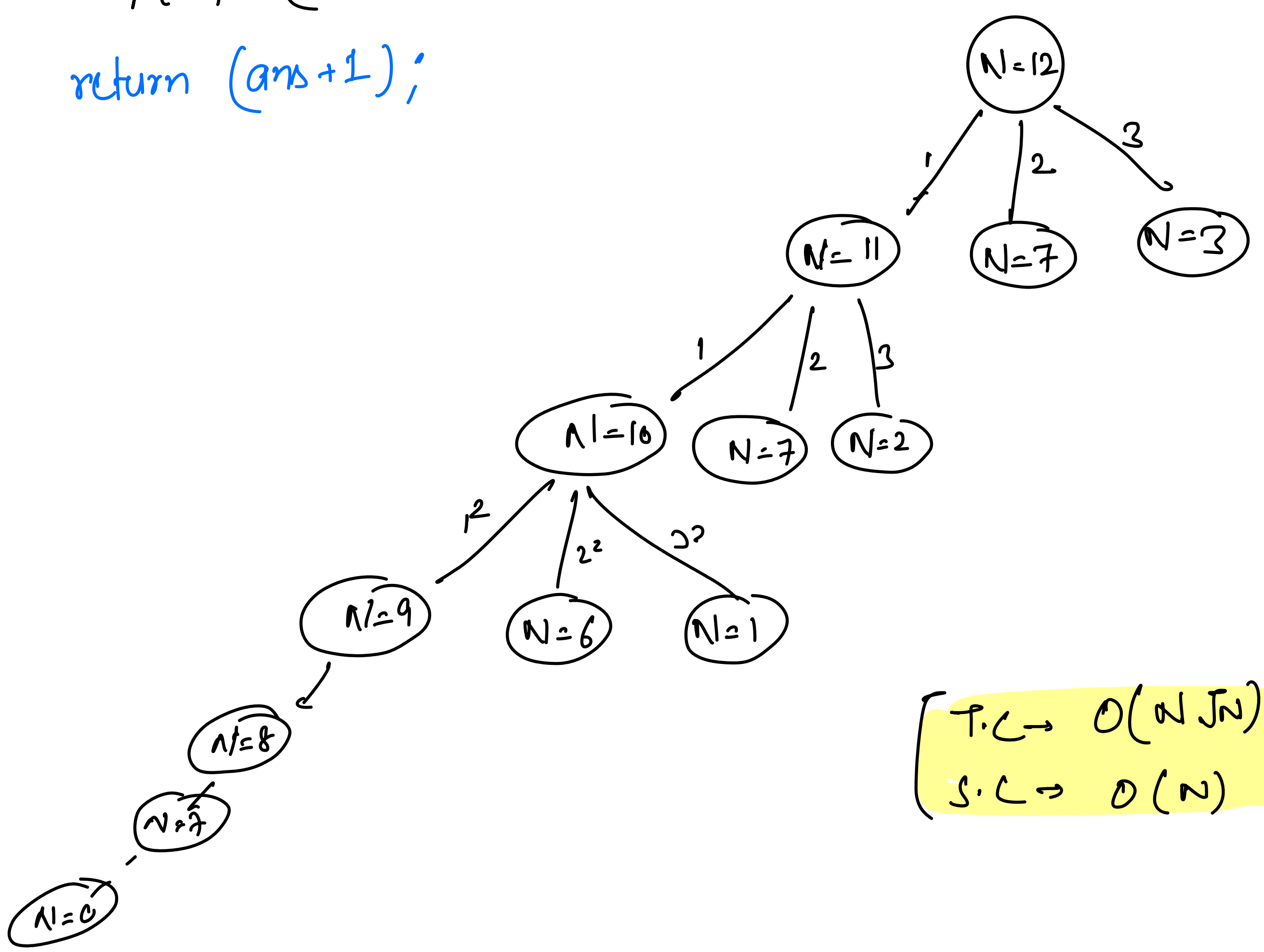
⇓

$$\left\{ minPSq(N) = \underset{\forall x^2 \le N}{Min}\left[minPSq(N-x^2)\right] + 1 \right\}$$

code:-

dp[N+1], $\forall i$, dp[i] = -1

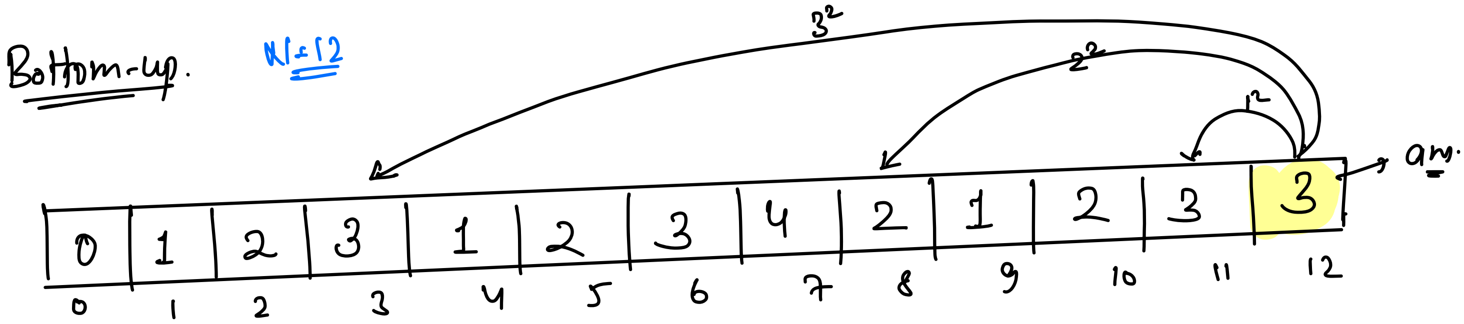
```
int minpsq( int N , int[] dp){
    if (N==0) {return 0}
    if (dp[N] != -1) { return dp[N]; }
    ans = ∞
    for( int x = 1 ; x * x <= N ; x++){
        ans = Min( ans , minpsq( N - x², dp));
    }
    dp[N] = (ans+1);
    return (ans+1);
}
```

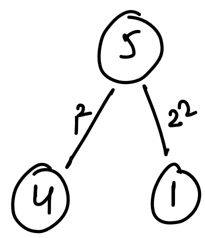


T.C → $O(N\sqrt{N})$

S.C → $O(N)$

Bottom-up. $N=12$





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 - x²]);
        }
        dp[i] = ans + 1;
}
return dp[N];
```

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

Max Sum of non-adjacent elements in an array →

7	3	5	9	2

16

ans = 16

7	5	3	9	2

× ✓

0 + maxSum(arr, N-1)

2 + maxSum(arr, N-2)

1 → decide the storage.

2 → store the ans in the storage before returning

3 → Before making the recursive call check if the answer is pre-calculated or not.

Eclipse