DP 1

Good Evening

Today's content

01. Introduction
02. Fibonacii
03. Stair ways (Amazon, Google)
04. Minimum squares to reach N (Walmart, Amazon)
05. Min no. of notes

Mock Interview

DSA Module → Skill Evaluation Contest → Mock Interview

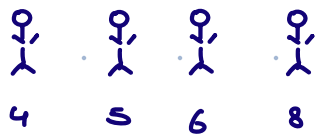
3-4 attempts

→ 5 questions

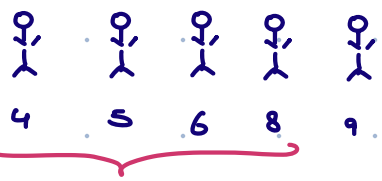
DSA certified

Dynamic Programming

Marks of students



⇒ Total = 23



⇒ Total = 23 + 9 = 32

→ Idea of DP → Use precalculated value to calculate current value & not calculate same thing again & again

* Prefix sum

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

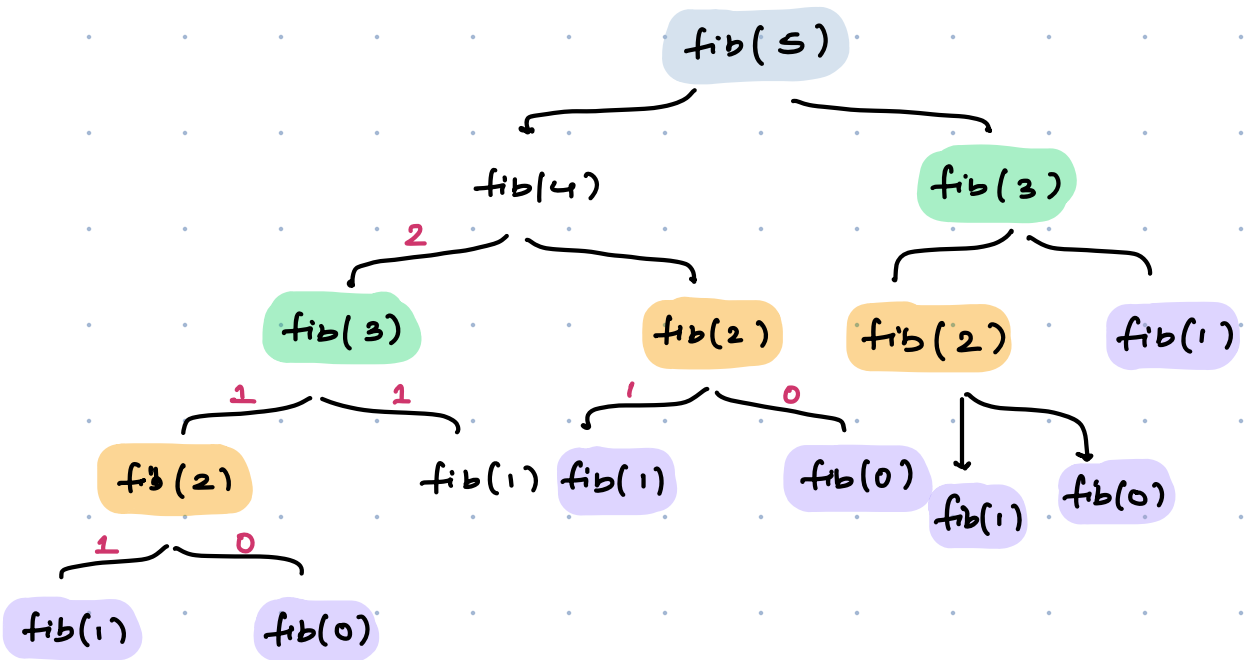
Fibonacci series

N =	0	1	2	3	4	5	6
fib =	0	1	1	2	3	5	8

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

TC: $O(2^n)$

SC: $O(N)$



Optimisation → Use DP

Store the result of already solved problem in an array & reuse it.

To apply DP

01. **Optimal substructure** → Solving a problem by using a smaller instance of problem.

02. **Overlapping subproblems** → Solving a subproblem again & again

```
int [] dp = new int [N+1];
Arrays.fill (dp, -1);
int fib( N , dp){
    if (N<=1) return dp[N]=N;
    if (dp[N] != -1) return dp[N];
    dp[N] = fib(N-1, dp) + fib(N-2, dp);
    return dp[N];
}
```

* Steps to Apply DP

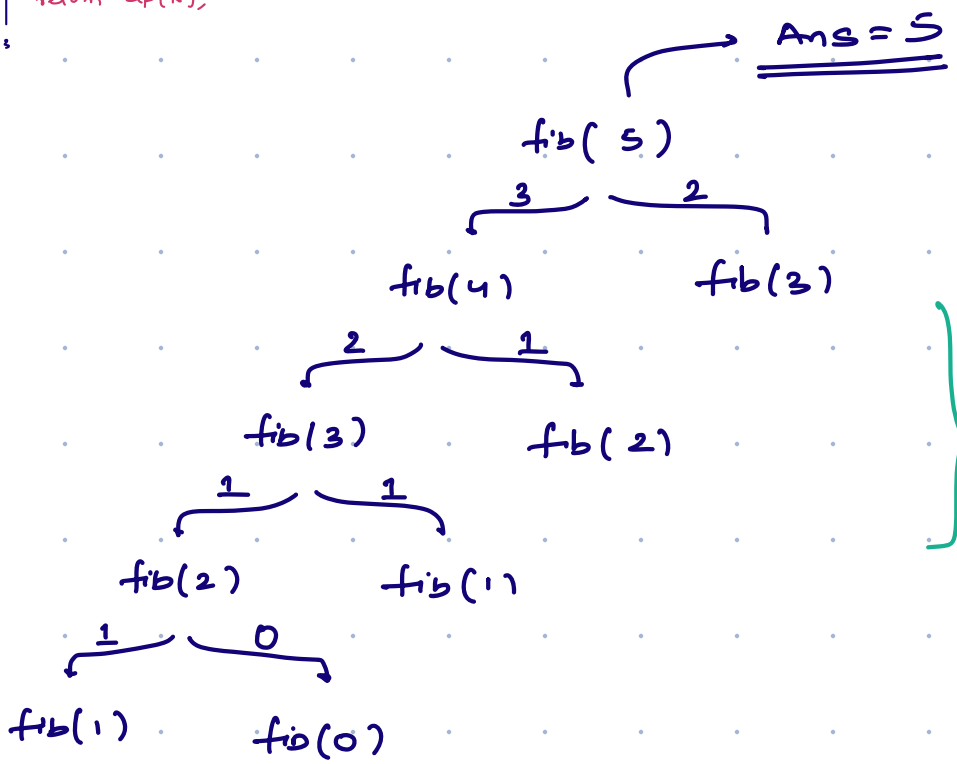
01. Initialise array with correct size to store subproblem answer.

02. Before making a recursive call, check if subproblem answer is already stored.

03. Before returning answer, store it in DP.

```
int [] dp = new int [N+1];
Arrays.fill (dp, -1);
int fib( N , dp){
    if (N≤1) return N;
    if (dp[N] != -1) return dp[N];
    dp[N] = fib(N-1) + fib(N-2);
    return dp[N];
}
```

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



TC: O(N)
SC: O(N)

sc : 2n

dp[] + recursive stack space

* Two types of DP

01. Memoisation / Top down Approach
    - → start from the biggest problem
    - → Recursive solution

02. Tabulation / Bottom Up Approach
    - → Start from the smallest problem (Base condition)
    - → Iterative solution.

```
int [] dp = new int [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];
```

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

TC : O(N)

SC : O(N)

*
```
a=0 , b=1

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

System.out.println( b );
```

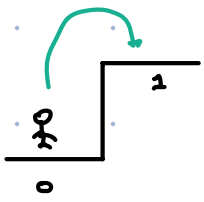
TC : O(N)

SC : O(1)

* Climbing Stairs

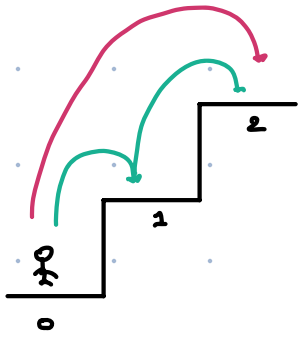
Calculate the no. of ways to reach N^{th} stair.

You can take a step of 1 or step of 2



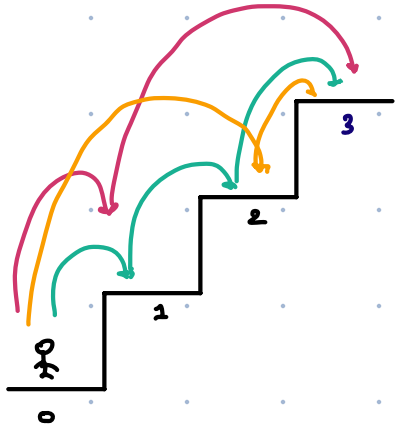
N = 1

Ans = 1



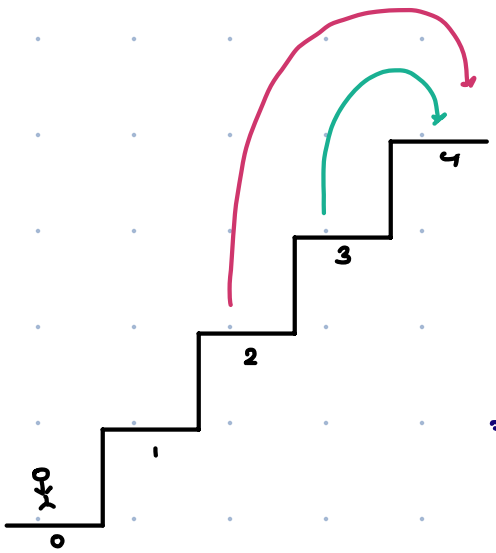
N = 2

Ans = 2 { 1 1 , 2 }



N = 3

Ans = 3 { 1 1 1 , 1 2 , 2 1 }



Ans = 5

{ 1 1 1 1 , 1 1 2 , 1 2 1 , 2 2 , 2 1 1 }

ways(4)

ways(3) + ways(2)

{ 1 1 1 1 , 1 2 1 , 2 1 1 }

{ 1 1 2 , 2 2 }

ways(4) = ways(3) + ways(2)

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

code by yourself

* Min cost to climb stairs → HW

* Minimum squares / Perfect squares

Find minimum no. of perfect squares required to get sum = N (duplicate squares are also allowed)

N = 6

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

$1^2 + 1^2 + 2^2 = 3$

Ans = 3

N = 10

$1^2 + 1^2 + 1^2 \ldots + 1^2 = 10$

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

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

$2^2 + 1^2 + 1^2 + 1^2 + 1^2 \Rightarrow 7$

$1^2 + 1^2$

Ans = 2

N = 5 $1^2 + 1^2 + 1^2 + 1^2 + 1^2 = 5$

$2^2 + 1^2 = 2$

Ans = 2

* Greedy Approach (N – nearest perfect square) ✗

10 → -9 → 1 → -1 → 0

Ans = 2

9 → -9 → 0

Ans = 1

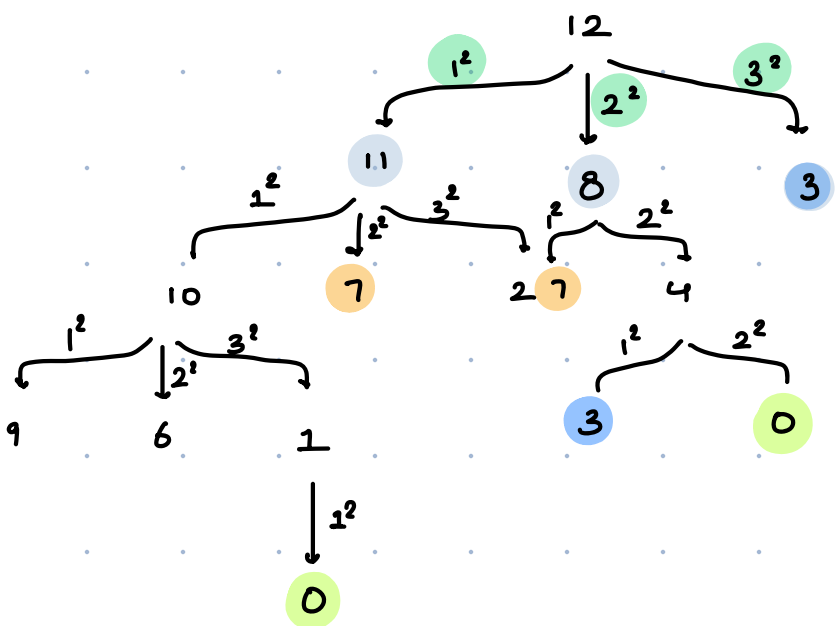
12 → -9 → 3 → -1 → 2 → -1 → 1 → -1 → 0

Ans = 4

12 → -4 → 8 → -4 → 4 → -4 → 0

Ans = 3

Solution → Explore every possible no.



TC : $O(\sqrt{n}^n)$

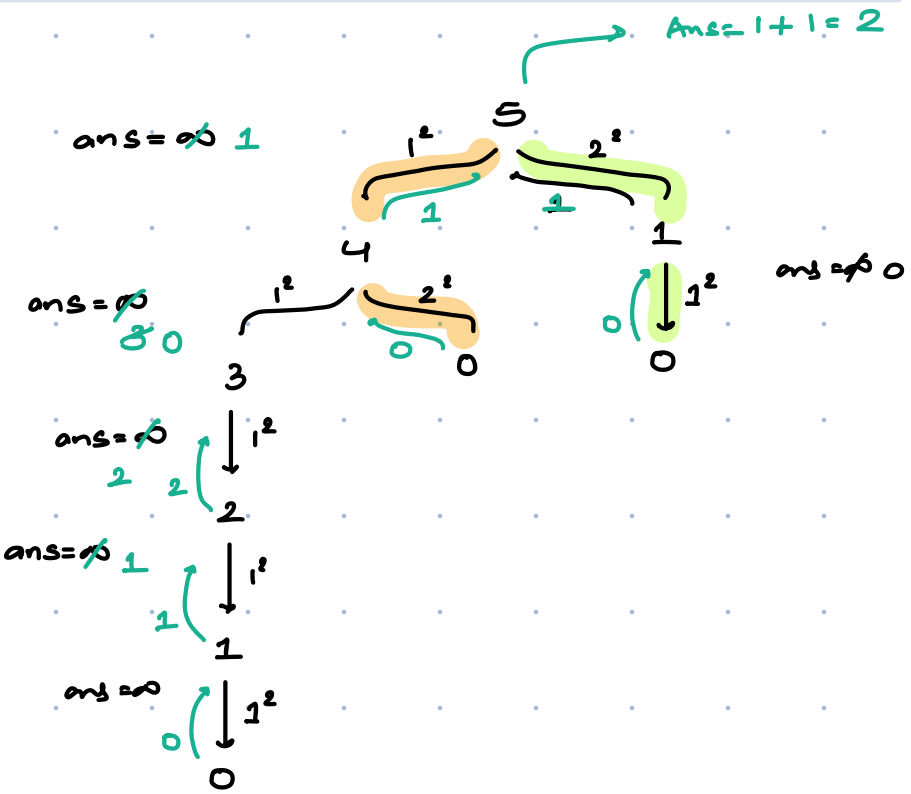
SC : $O(n)$

$$minpsq(12) = Min\left(minpsq(11), minpsq(8), minpsq(3) \right) + 1$$

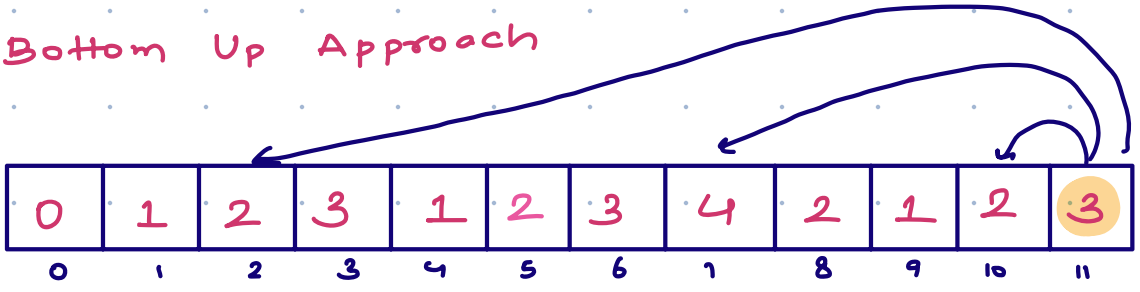
$12-1^2$ $12-2^2$ $12-3^2$

$$minpsq(N) = Min\left[Minpsq(N-x^2) \text{ where } x^2 \le N \right] + 1$$

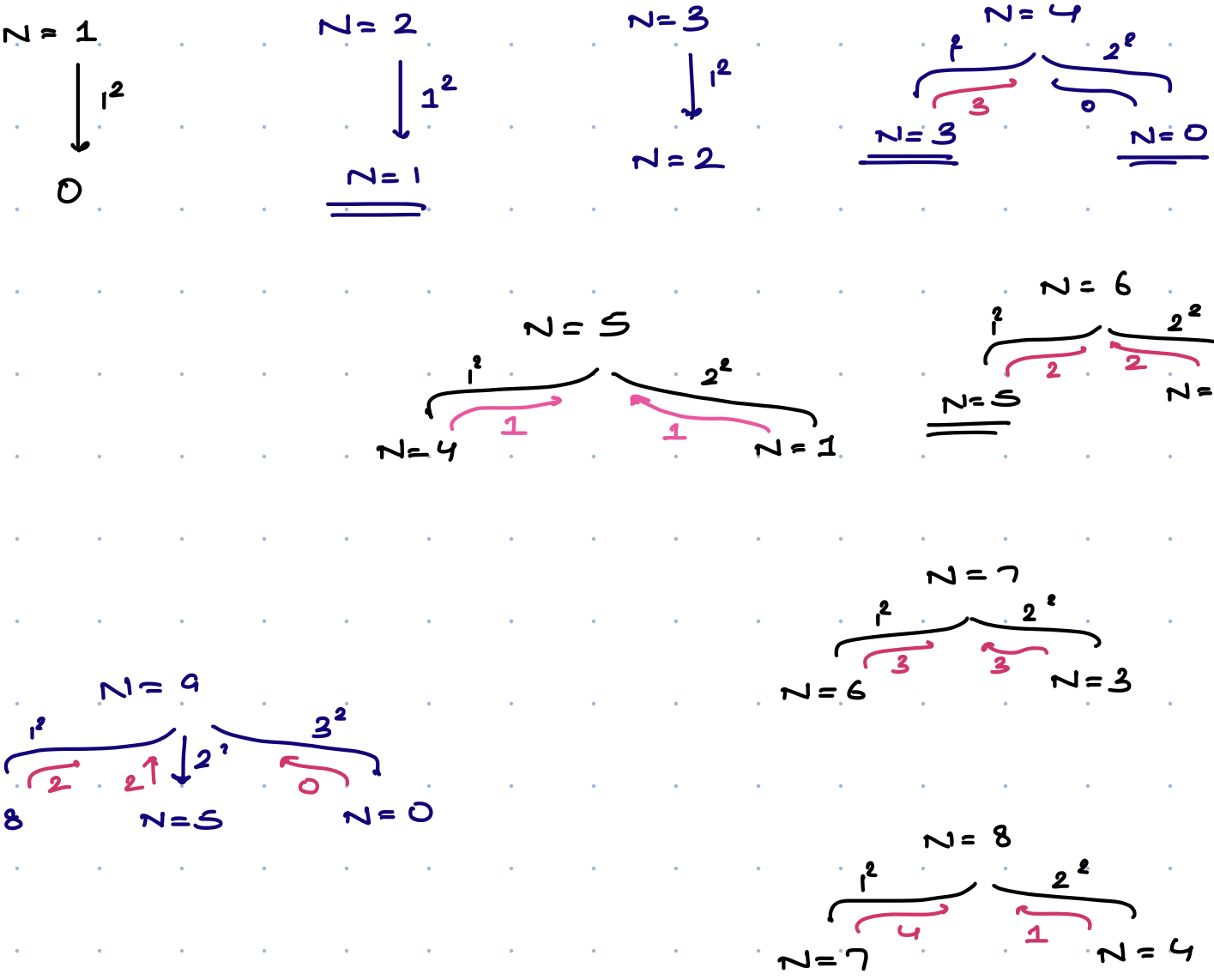
```
int [ ] dp = new int [N+1]
Arrays.fill(dp, -1);
int minpsq ( N )
    if (N==0) return 0
    if (dp[N] != -1) return dp[N];
    ans = ∞
    for (x=1 ; x*x ≤ N ; x++){
        ans = Min (ans, minpsq(N - x²));
    }
    return dp[N] = ans + 1;
}
```



* Bottom Up Approach



dp(i) = min no. of squares required to make N



```
int [] dp = new int [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];
```

TC: $O(N\sqrt{N})$

SC: $O(N)$