- What is Dynamic Programming?
- Conditions to use DP
- Why DP? → Fibonacci Series
- No. of Stairs
- Min. Perfect Squares

Contest 5 → Next Fri

Contest 3 and 4 → Reattempt

DP

Graphs

Jan End

↓

DSA Mock Interview

Fibonacci Series

N	0	1	2	3	4	5	6	.....
	0	1	1	2	3	5	8	......

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

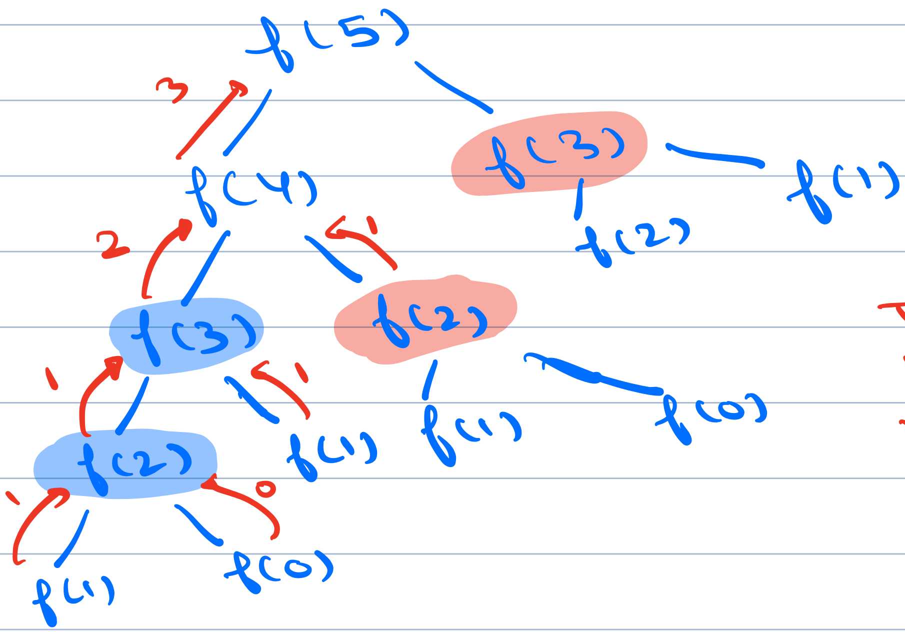
```
int fib(n) {
    if (n <= 1)              <- Base Case
        return n

    return fib(n-1) + fib(n-2)   <- Recursive relation
}
```

→ TC : $O(2^N)$

SC : $O(N)$

$N = 10$ Iterations → $2^{10} = 1024 = 10^3$

$N = 20$ Iterations → $2^{20} = 10^6$



Too many repetitive calls

DP → when some problems repeat again, store their ans

Conditions for DP

1. Optimal substructure : solving a problem by breaking into similar subproblems
2. Overlapping subproblems

N

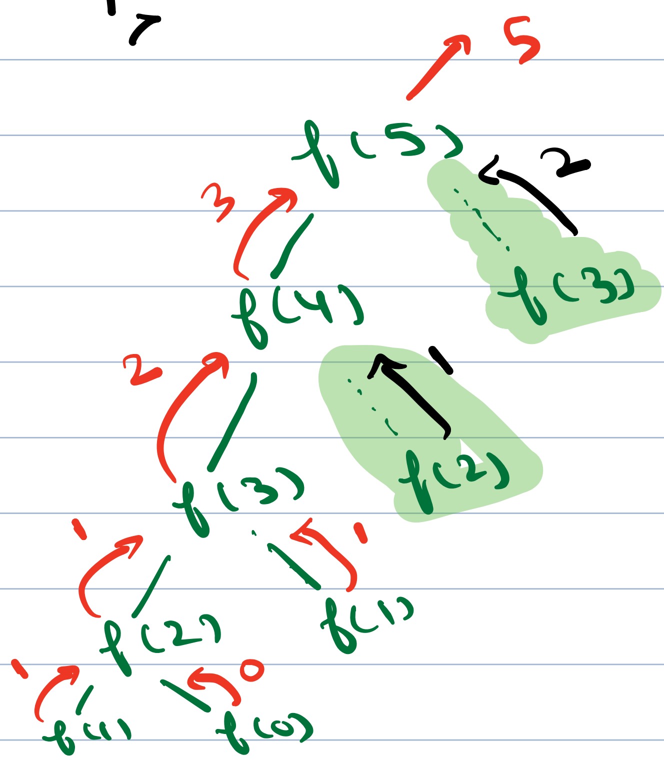
N-1

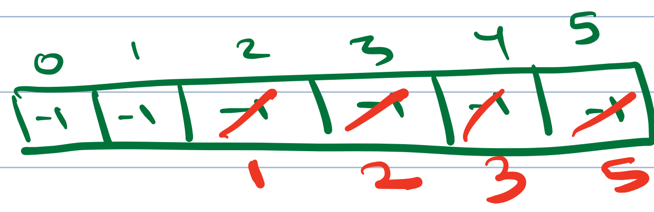
N-2

⋮

0

```
int dp[N+1] = {-1}

int fib(n) {
    if (n <= 1)
        return n
    if (dp[N] != -1)
        return dp[N]

    dp[N] = fib(n-1) + fib(n-2)
    return dp[N]
}
```



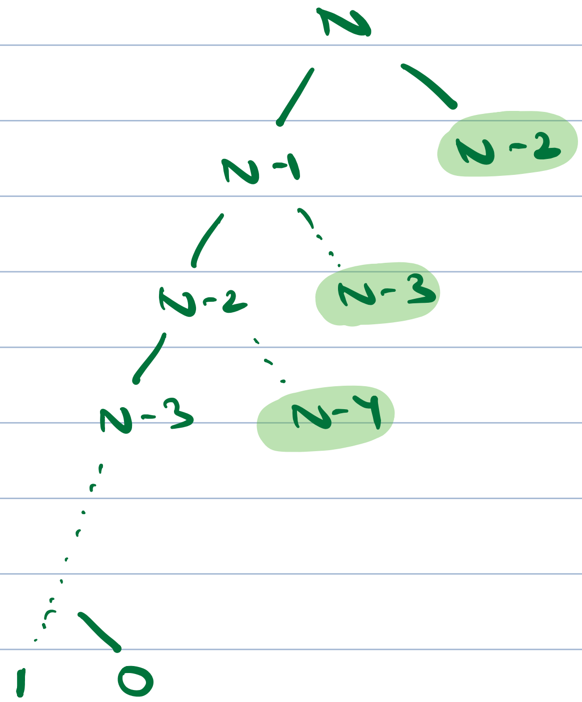


TC : O(N)

SC : O(N + N)

↓

O(N)



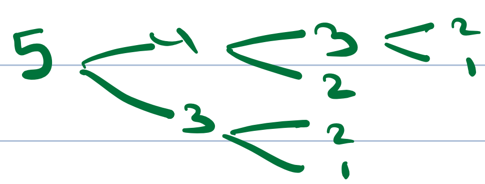
fn (N) {

Base case

if (ans for N already stored)
return from memory

memory = Recursive call
return memory

}



Types of DP

Recursion	Iterative
Top Down	Bottom Up
(Memoization)	(Tabulation)

5 → 4 → 3 → 2 → 1   0

```
dp[N+1] = {-1}

dp[0] = 0
dp[1] = 1

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

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

N=5

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

f(3)
f(2)

* Bottom up DP with SC: O(1) // $0^{th} \rightarrow N^{th}$ term

```
a = 0
b = 1

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

return c
```

i=2

a	b	c
0	1	1

i=3

a	b	c
1	1	2

i=4

a	b	c
1	2	3

a	b	c
0	1	1
	a	b

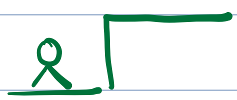
TC : O(N)

SC : O(1)

---

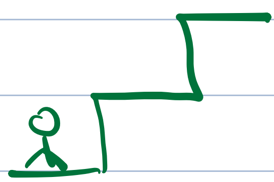
2. Given N stairs, in how many ways we can go from 0th to Nth stair if we take a jump of 1 stair or 2 stairs at a time?

N = 1



<1> ans = 1

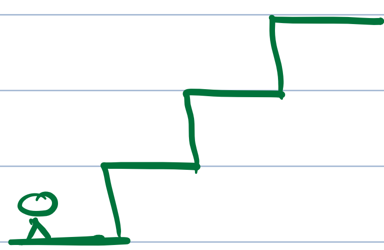
N = 2



<1, 1>
<2>

ans = 2

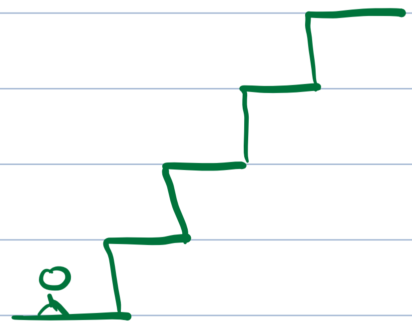
N = 3



<1, 1, 1>
<1, 2>
<2, 1>

ans = 3

$N = 4$



$\langle 1, 1, 1, 1 \rangle$

$\langle 1, 1, 2 \rangle$

$\langle 1, 2, 1 \rangle$

$\langle 2, 1, 1 \rangle$

$\langle 2, 2 \rangle$

ans = 5

4th step $\Rightarrow$ 5

1 stair — 3rd step

$\langle 1, 1, 1 \rangle$ 1

$\langle 1, 2 \rangle$ 1

$\langle 2, 1 \rangle$ 1

2 stairs — 2nd step

$\langle 1, 1 \rangle$, 2

$\langle 2 \rangle$, 2

ways (N^{th} step)

1 stair — ways($N-1^{th}$)

1

1

1

2 stairs — ways($N-2^{th}$)

2

2

2

2

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

5th step → 8

1 2

5 4th step 3rd step 3

4th step		3rd step	
<1,1,1,1>	1	<1,1,1>	2
<1,1,2>	1	<1,2>	2
<1,2,1>	1	<2,1>	2
<2,1,1>	1		
<2,2>	1		

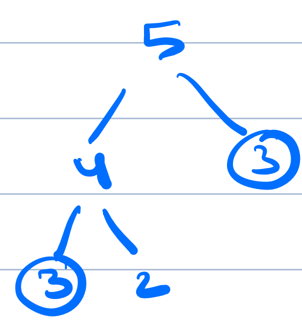
① DP relation

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

② Base Case

N=1 ways(1) = 1

N=0 ways(0) = 1 ↓ Do nothing



ways(2) = ways(1) + ways(0)

↓ 2 ↓ 1 ↓ 1

ways(-5) = 0 No ways

N = 2 ways(2) = 2

3. Find minimum number of perfect squares required to get sum = N

$1, 4, 9, 16, 25, \dots$

		cnt
N = 2	$1^2 + 1^2$	2
N = 3	$1^2 + 1^2 + 1^2$	3
N = 4	$1^2 + 1^2 + 1^2 + 1^2$ 2^2	1
N = 5	$1^2 + 1^2 + 1^2 + 1^2 + 1^2$ $2^2 + 1^2$	2

✗ Greedy Approach → To make no. n, use biggest perfect square possible

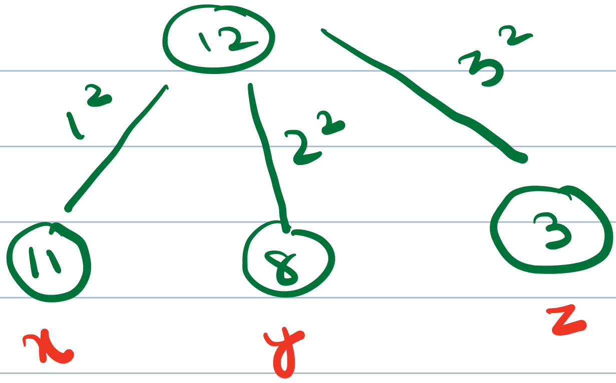
$$12 \xrightarrow{-3^2} 3 \xrightarrow{-1^2} 2 \xrightarrow{-1^2} 1 \xrightarrow{-1^2} 0$$

greedy → 4 terms

$$2^2 + 2^2 + 2^2$$

actual ans = 3

Brute Force: Try every possible way to form sum

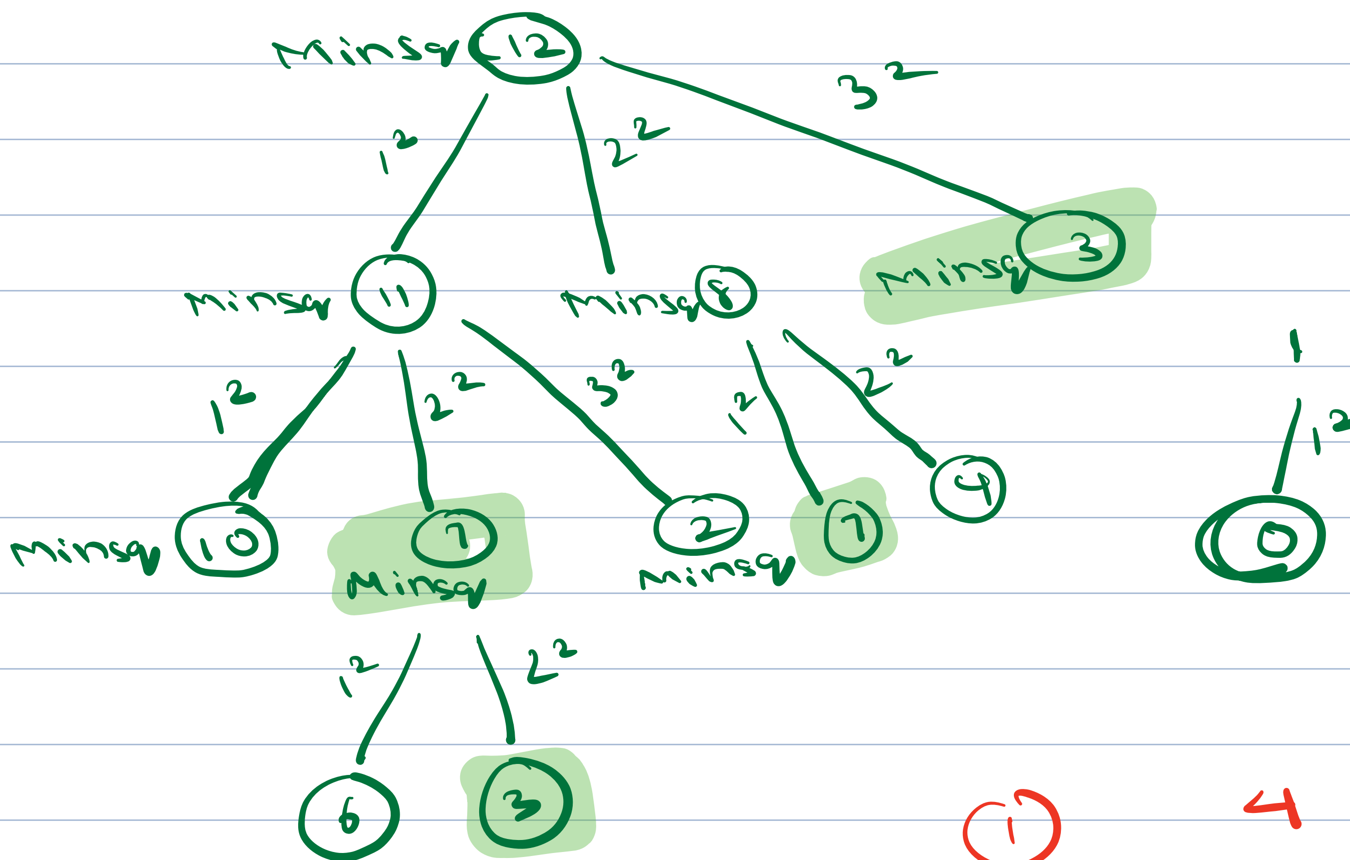


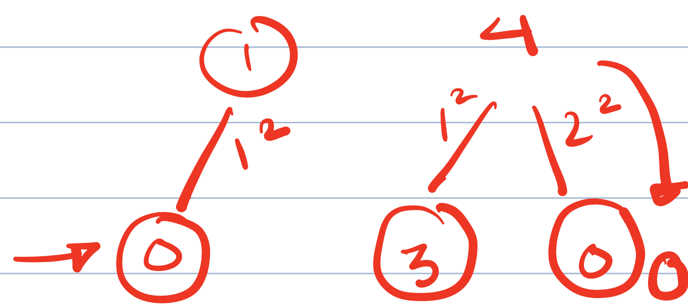
Minsq(12)

↓

min(x, y, z) + 1

$$\text{Minsq}(12) = \text{Min}(\text{Minsq}(11), \text{Minsq}(8), \text{Minsq}(3)) + 1$$

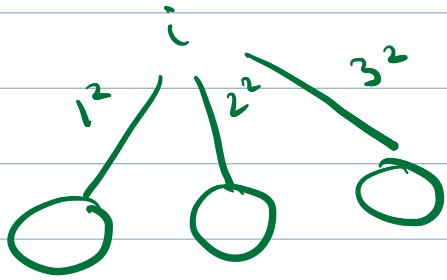


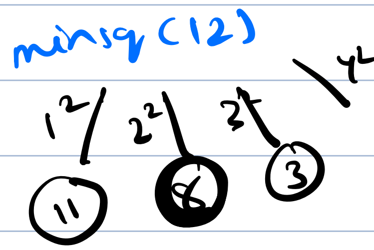


$Minsq(i) = \min\{Minsq(i - x^2)\} + 1$

for all $x \Rightarrow x^2 \le i$

$MinSq(0) = 0$






```
int dp[N+1] = {-1}

int minsq(int N) {
    if (N == 0)
        return 0

    if (dp[N] != -1)
        return dp[N]

    int minval = INT_MAX
    for( x = 1 ; x * x <= N ; x++) {
        minval = min(minval, minsq(N - x^2))
    }

    dp[N] = minval + 1
    return dp[N]
}
```

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

SC: $O(N)$

Iterative

0 → 1 → 2 → 3

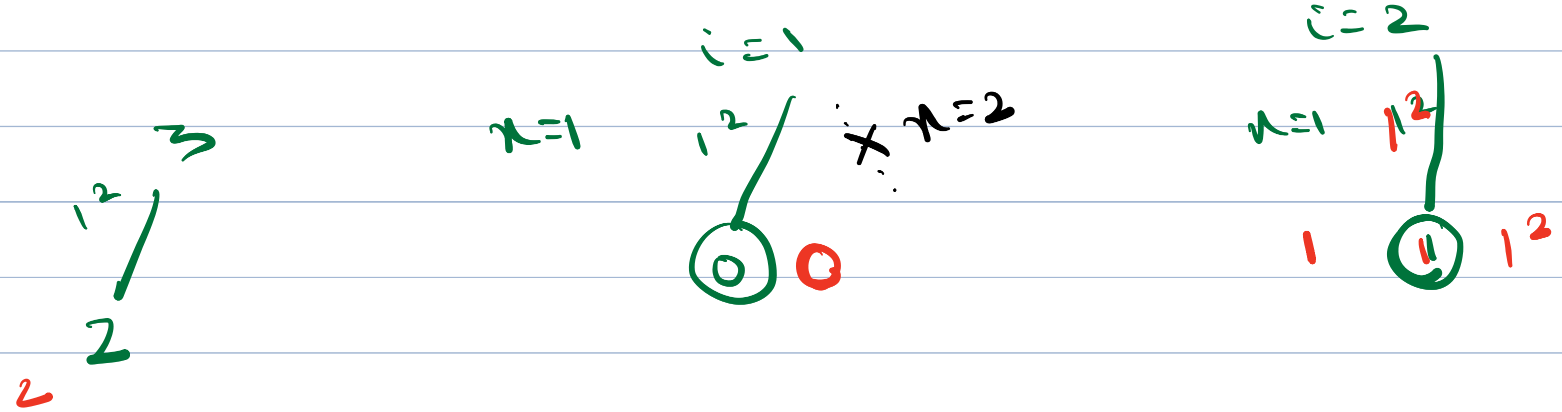
```
int dp[N+1] = {-1}
dp[0] = 0

for (i=1 ; i <= N ; i++) {
    int minval = INT_MAX
    for (x=1 ; x*x <= i ; x++) {
        minval = min(minval, dp[i - x^2])
    }
    dp[i] = minval + 1
}
```

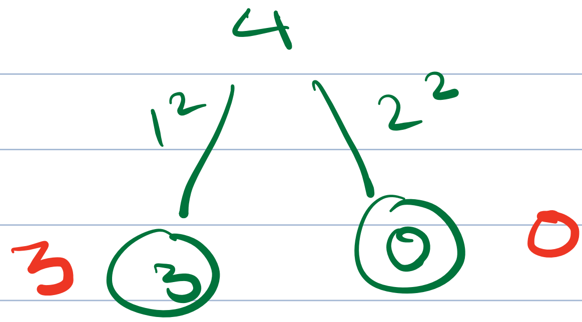
TC : $O(N\sqrt{N})$
SC : $O(N)$

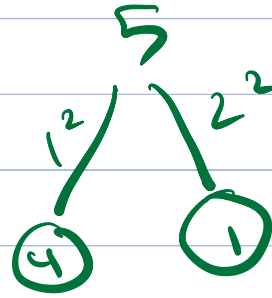
dp[i] = Minsq to make i

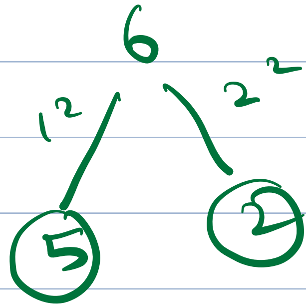
$N = 9$

0	1	2	3	4	5	6	7	8	9
0	1	2	3	1					




4
1²
2²
3
0
3
0


5
1²
2²
4
1


6
1²
2²
5
2