Q Q Q Q Q Q

3 4 1 2 5 6 } 21 Q 10 $\underline{n^2}$

21 + 10 = 31 + 5 = 36

{ using previously found value } n

	3	4	1	2	5	6	
pf	3	7	8	10	15	21	...

$pf[i] = a[i] + pf[i-1]$

Fibonacci series :-

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

$$\boxed{fib(n) = fib(n-1) + fib(n-2)}$$

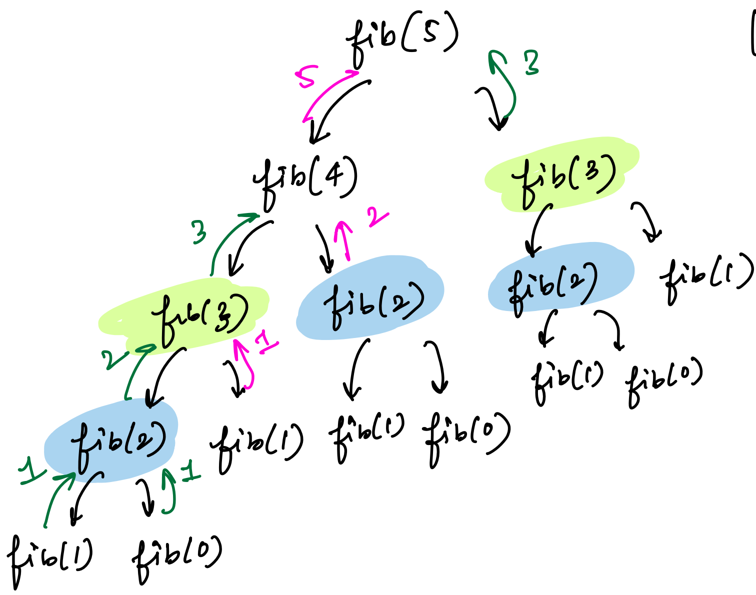
$fib(0) = 1 \quad fib(1) = 1$

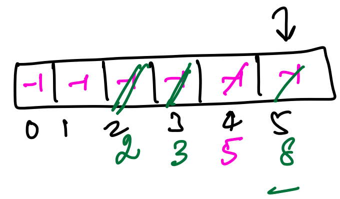
```
int fib(int n)
{
    if( n <= 1) return 1;

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

T.C : $O(2^n)$
S.C : $O(n)$

optimal substructure :- solve big problems using smaller subproblems.





overlapping subproblems :- when you have same subproblems multiple times

store subproblems result to avoid repetition

int fib[n+1] → fill this array with -1

fib[0]=1

fib[1]=1

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

T.C: O(n)

S.C: O(n)

n → storage

n → recursion call

top down / recursive → start from the biggest

memoisation :- storage of results and using the data to solve further

Bottom-up / iterative :- tabulation

```
int fib[n+1]; ⟹ fib[0] = 1, fib[1] = 1
               fib[2] = fib[1] + fib[0]
               fib[3] = . . .

for (int i = 2; i <= n; i++)
{
      fib[i] = fib[i-1] + fib[i-2];
}
```

T.C : O(n)

S.C : O(n) but without recursive stack

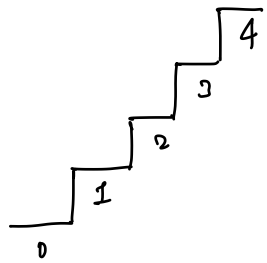
```
int a = 1;
int b = 1;
      int c;
for ( int i = 2; i <= n; i++)
{
      c = a + b;
      a = b;
      b = c;
}
```

~~a=1~~ b=1 ~~c=2~~ c=3
  ↑   ↑   ↑
 ~~a~~   ~~b~~   b
         a

S.C: O(1)

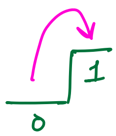
Stairs

N^{th} stair $\equiv$ How many no of ways to reach N^{th} stair!



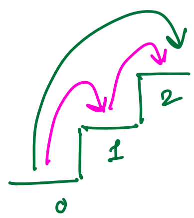
1 step = 1 stair / 2 stair

N=1



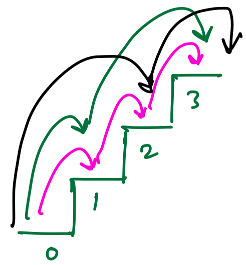
ans = 1

N=2



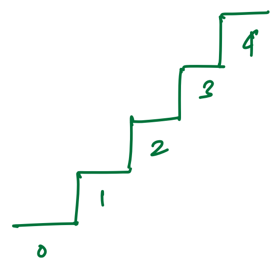
ans = 2

N=3



ans = 3

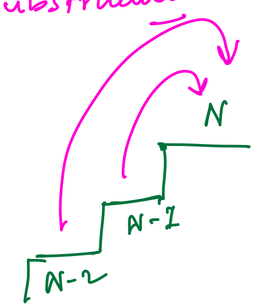
N=4

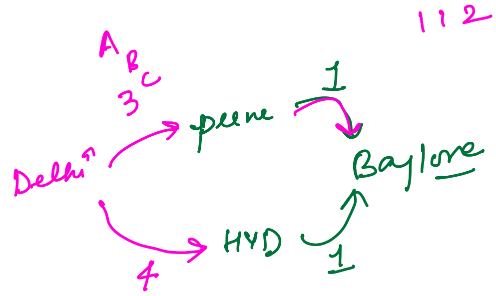


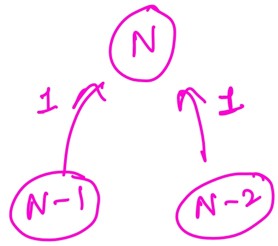
ans = 5

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

optimal substructure







Base case :- the values which you can't find with eqⁿ + stopping step

$3*1 + 4*1 = 7$

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

$ways(1) = 1$ and $ways(2) = 2$

$ways(0) = 1$

#3 Find min no of perfect squares needed get sum = N.

{greedy}

$N = 6 \quad = 4 + 1 + 1$
$(2^2) \quad (1^2) \quad (1^2)$

$1 + 1 + 1 + 1 + 1 + 1$

N − (largest perfect square <= N)

$N = 10$

1 . . . 10 times

$4 + 4 + 1 + 1$

$9 + 1$

$4 + \ldots$

$N = 12$

$12 - 9 = 3$

$3 = 1 + 1 + 1$

$4 + 4 + 4 = 12$

$N = 9$

1 . . . 9 times

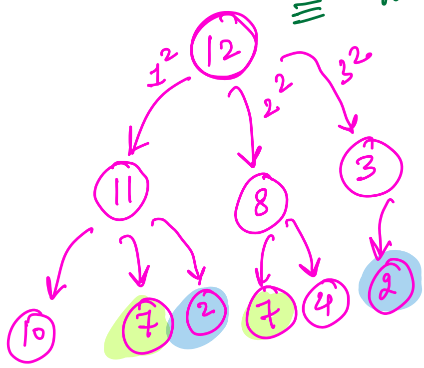
$4 + 4 + 1$

9

$\boxed{\text{element of choice}}$

optimal substructure

overlapping subpr.



$\equiv$ min no of square to reach 12

$dp[i]$ = min no of square to get a sum of i

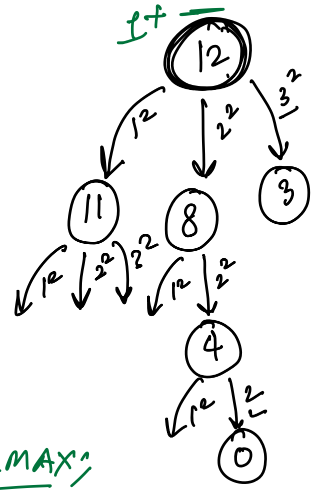
$dp[12] = 1 + \min(dp[11], dp[8], dp[3])$

$dp[12-1^2], dp[12-2^2], dp[12-3^2])$

$$\boxed{dp[i] = 1 + \min_{\substack{\forall x, \; x^2 <= i \\ x >= 1}} dp[i - x^2]}$$

$dp[9] = 1 + \min(dp[9-1], dp[9-4], dp[9-9]);$

$dp[0] = 0$




```
int dp[n+1]    // -1

int psquares( int n)
{
        if( n==0) return 0;
        if(dp[n] != -1) return dp[n];
         dp[n]=1;    int ans = INT_MAX;
        for( x=1; x*x<=n; x++)
        {
                ans = min( ans, psquare(n-x^2));
        }
        dp[n] += ans;
        return dp[n];
}
```

T.C: $O(n\sqrt{n})$ S.C: $O(n)$

1 2 3 4 .. 12

```
for( i=1; i<=n; i++)
{
        for( x=1; x*x <= i; x++)
        {
        }
}
```

4

roll a dice $\{1 \rightarrow 6\}$, no of ways in which you can get sum = N!

12

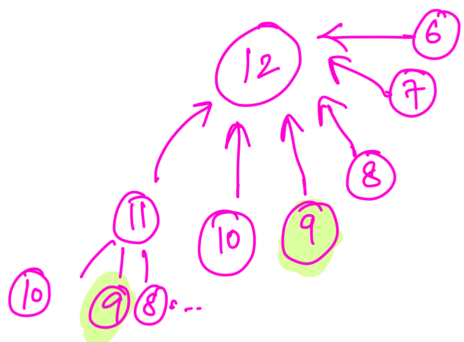
6 + 6

1 + 1 + 1 .. 12 times

5 + 4 + 3

6 + 4 + 2

⋮



$$dp[n] = dp[n-1] + dp[n-2] + dp[n-3] + dp[n-4] + dp[n-5] + dp[n-6]$$

$$\boxed{dp[n] = \sum_{j=1}^{j<=6,\ n>=j} dp[n-j]}$$

$dp[0] = 1$!

~~$dp[1] = 1/6$~~

$dp[2] = 2$;

$dp[3] = 4$ $\quad dp[1] + dp[2] + 1$

$dp[4] = 8$

$dp[5] = 16$

$dp[6] = 32$;