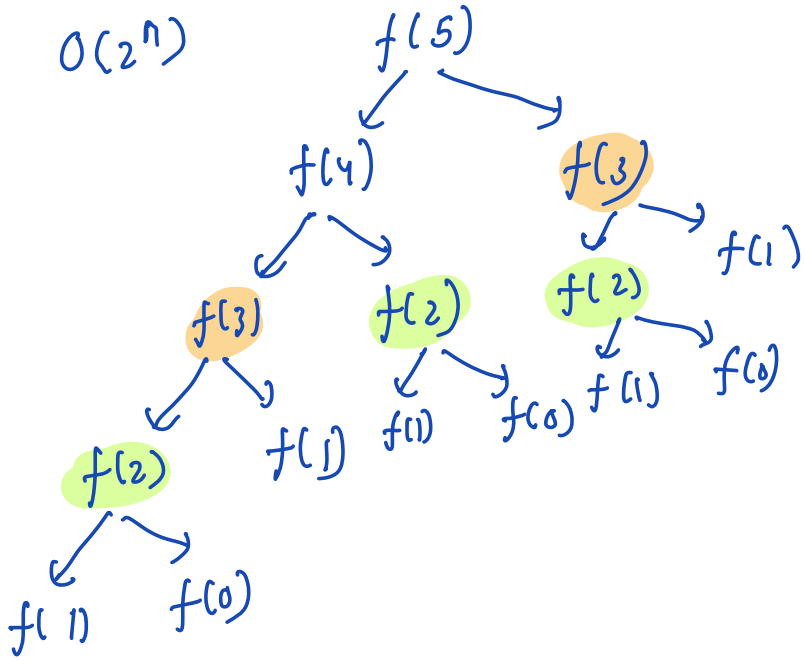
Dynamic Programing 1

0	1	2	3	4	5		
0	1	1	2	3	5	8	13

```
int fib( int N)
{
  if (N <=1) return N

  return { fib( N-1) +
           fib( N-2) }
```

$O(2^n)$



① Solving a problem with subproblem
: optimal substructure

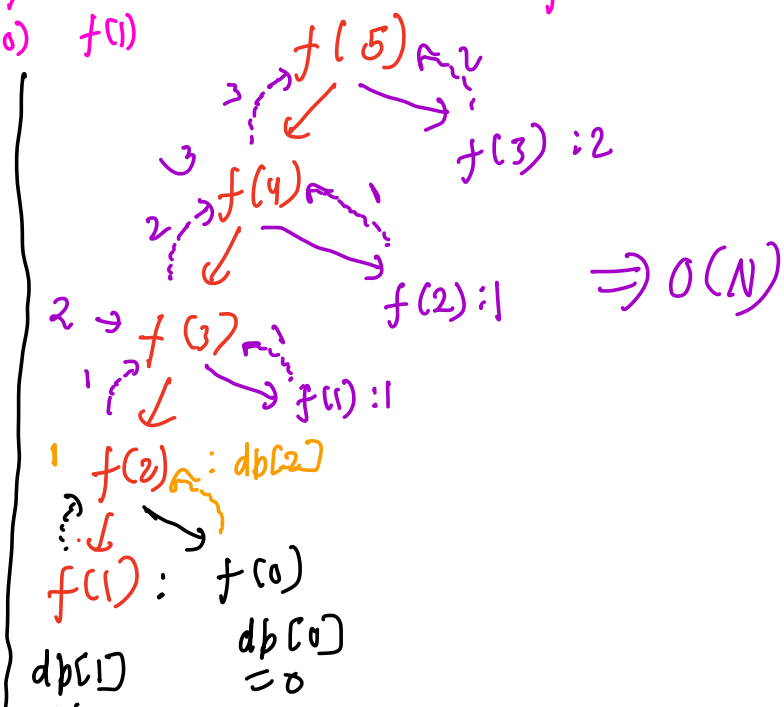
② solving same problem more than once
: overlapping subproblem

↳ By calling each unique subproblem once : DP

db[6] =

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

f(0) f(1) f(5)



⇒ O(N)

```
int db[N+1]

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

Top to bottom approach

$\boxed{\text{Recursion + Memory} \Rightarrow \text{Memoization}}$

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

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

$dp(0) \rightarrow dp(1)$
$\downarrow$
$dp(2)$
$\downarrow$
$dp(3)$

```
int fibite (int N)
{
    int dp[N+1];
    dp[0] = 0 , d[1] = 1

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

Bottom up Approach

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

⇒

```
int fib2 (int n)
{
    int a = 0 ; b = 1;
    int c;
for( int i = 2; i <= N; i++)
    {
```

a	b	c
0	1	1 → 2nd
1	1	2 → 3rd
1	2	3 → 4th
2	3	5

0	1	2	3	4	5

0	1	1	2	3	5

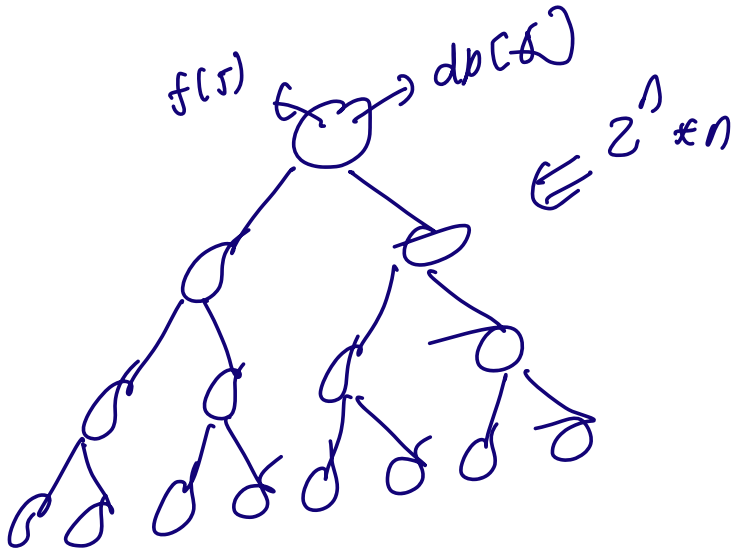
```
    c = a+b;
    a = b;
    b = c;
}
return c;
}
```

$O(N)$: TC

$O(1)$: SC

TC: { # of dp states * TC at each state }

SC: { dp table size }

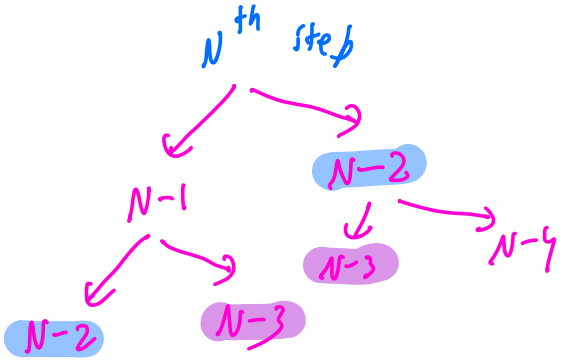


Q2 ⇒ Given N stairs, how many ways we can go from 0 → N^{th} step.

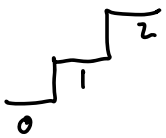
Note: From i^{th} step we can directly go to (i+1) or (i+2) steps

Ex: N=1

: way: 1



N = 2

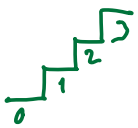


: ways : 2

1 1

2

N = 3



: ways : 3

1 1 1

2 1

1 2

N = 4

: ways : 5

1 1 1 1

1 1 2

1 2 1

211

22

N = 4

1111 → 0 → 1 → 2 → 3 → 4

112 → 0 → 1 → 2 → 4

121 → 0 → 1 → 3 → 4

211 → 0 → 2 → 3 → 4

22 → 0 → 2 → 4

111, 12, 21 → +1 ⇒ 4

11, 2 → +2, +1 ⇒ 4

$$dp[i] = dp[i-1] + dp[i-2]$$

$dp[1] = 1$

$dp[2] = 2$

$$dp[2] = dp[1] + dp[0]$$

$dp[0] = 0$

$$= 1 + 0 = 1$$

$$dp[2] \Rightarrow dp[1] + dp[0]$$

$dp[0] = 1$

$$\Rightarrow 1 + 1$$

$$\Rightarrow 2$$

Break: 8:51

Q 5) 2-face $\begin{cases} 1 \\ 2 \end{cases}$, we can roll dice as many times as we want.

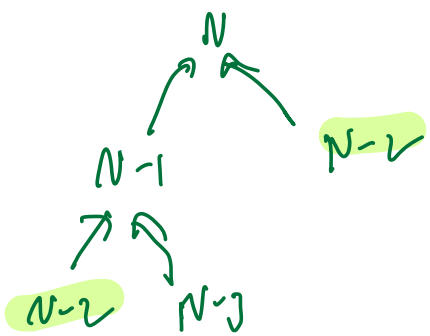
no of ways to get sum N?

N=1 : ways : 1 : {1}

N=2 : ways : 2 {1, 1}
{2}

N=3 : ways : 3 {1, 1, 1}
{1, 2}
{2, 1}

N=4 : ways : 5 : {1, 1, 1, 1}
: {1, 1, 2}
: {1, 2, 1}
: {2, 1, 1}
: {2, 2}



dp[i] = dp[i-1] + dp[i-2]

Q ⇒ 6-faced dice · we can throw $\{1, 2, 3, 4, 5, 6\}$

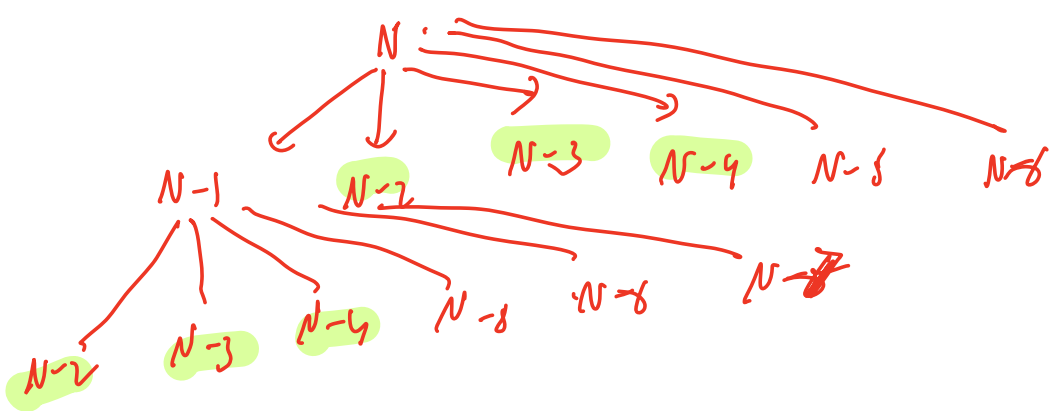
No of ways to get sum = N

$N=1$ ways : $\{1\}$

$N=2$ 2 : $\{1,1\}$
$\{2\}$

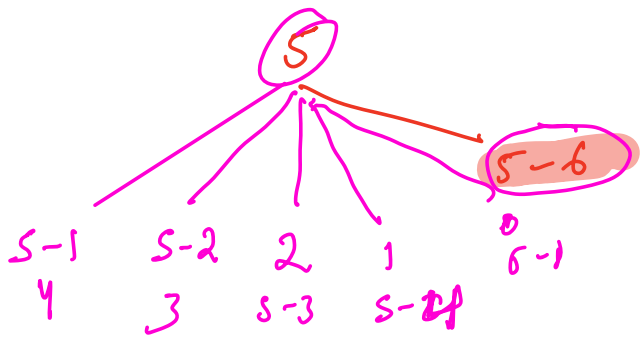
$N=3$ 4 : $\{1,1,1\}$
$\{1,2\}$
$\{2,1\}$
$\{3\}$

$N=4$ 8 : $\{1,1,1,1\}$
$\{4\}$ $\{2,1,1\}$ $\{1\,3\}$ $\{1\,1\,2\}$
$\{3\,1\}$ $\{2\,2\}$ $\{1\,2\,1\}$



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

$$dp[i] = \sum_{j=1}^{6} dp[i-j]$$



Modified dp :

$$dp[i] = \sum_{j=1}^{6} dp[i-j] \quad i \geq j$$

Edge case
$dp[0] = 1$

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

$dp[2] = dp[1] + dp[0] = 1 + 1 = 2$

```
int dp[N+1]
dp[0] = 1;
for (i = 1; i <= N; i++)
{
    int s = 0;
```

0	1	2	3	4
1	1	2	4	8

```
for( j=1 ; j<=6 && j<=i ; j++)
  {
    s = s + dp[i-j];
  }
  dp[i] = s;
}
return dp[n];
}
```

$N * 6$

TC: $(N) * (1)$
: $O(N)$

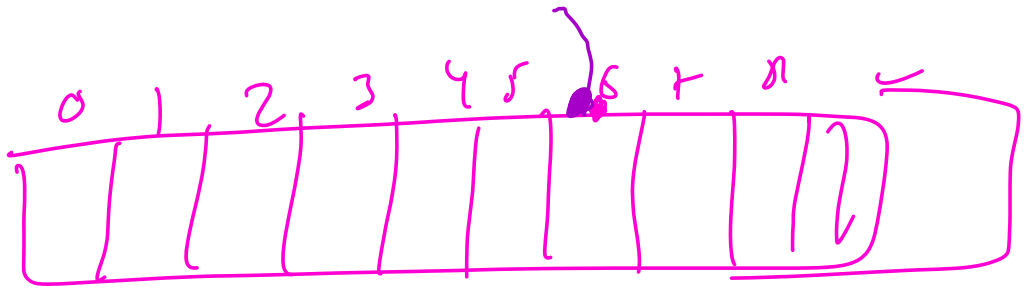
y

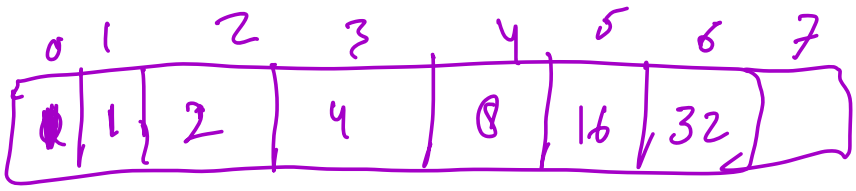
0	1	2	3	4
1	1	1	1	1

optimization → 7 way
⇓
TODO

N → 1 million → 6 million
N → 1 Billion → 1 Billion

→ A → sh → Mom






```
int dp[N+1]
dp[0] = 1;
for ( i = 1 ; i <= N; i++)
  {
      int s = 0;
      for( j=1 ; j <= 6 && j <= i ; j
        {
          s = s + dp[i-j];
        }
      dp[i] = s;
  }
  return dp[n];
```

0	1	2	3	
1	1	2	4	8

N

j=7

TC: (N)

: O(