Get the nth term

0 1 2 3 4 5 6 7 8 9 10 11

⇒ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 - - - -

2 → 1
3 → 2

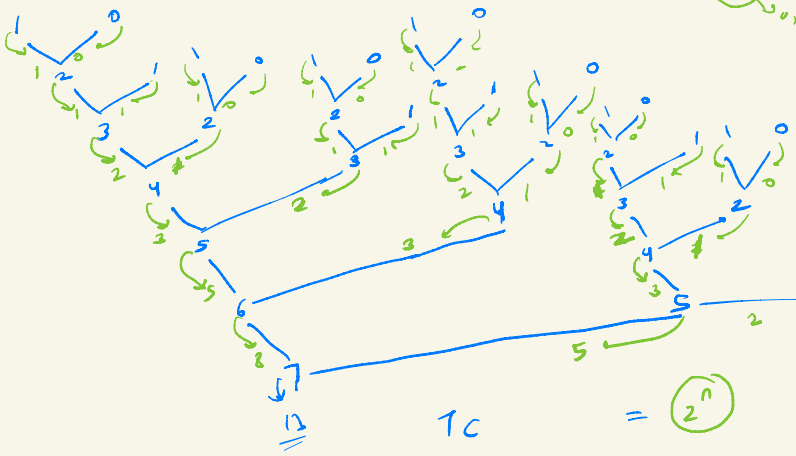
4 → 3 , 6 → 8

memoization

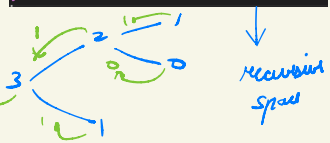
n+1

0, 1, 2, 3, 4, 5, 6, 7

n+1



```
public static int fib_rec(int n){
    if(n==0 || n==1){
        return n;
    }

    int ans = fib_rec(n-1) + fib_rec(n-2);

    return ans;
}
```



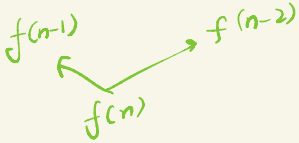
recursion space

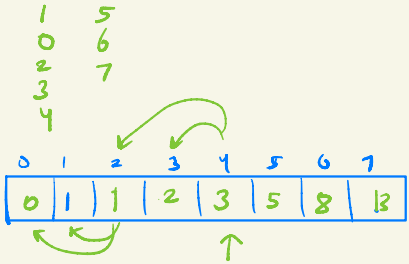
13

TC = 2^n

SC =

Overlapping subproblems

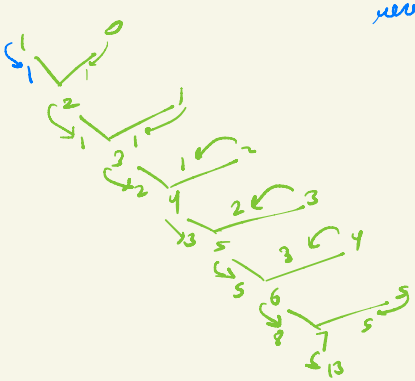




memo(4) = memo(3) + memo(2)

recursive space

O(n)



TC = O(n)

SC = O(n) + rec. space

```
public static int fib_memo(int n, int[] memo){
    if(n==0 || n==1){
        return memo[n]=n;
    }

    if(memo[n]!=0) return memo[n];

    int ans = fib_memo(n-1,memo) + fib_memo(n-2,memo);

    return memo[n]=ans;
}
```

1) Find the flow, write the for loop
2) Replace return with continue;
3) Remove memoization line (why?)
4) Replace calls with dp.

```java
public static int tab(int N, int[] memo){
    for(int n=0; n<=N; n++){
        if(n==0 || n==1){
            memo[n]=n;
            continue;
        }

        // if(memo[n]!=0) return memo[n];

        int ans = memo[n-1] + memo[n-2]; //fib_memo(n-1,memo) + fib_memo(n-2,memo)

        memo[n]=ans;
    }

    return memo[N];
}
```

n=~~0 1 2 3~~ 4 5

TC = O(n)
SC = O(n)

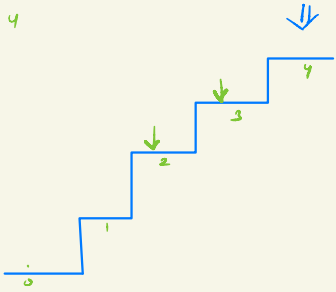
	0	1	2	3	4	5	6	7
memo =	0	1	1	2	3	5	8	13

```java
public static int tabulation(int n){
    int[] dp = new int[n+1];

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

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

    return dp[n];
}
```

Recursion ( )
↓
Memoization ( ) add two lines
↓
Tabulation ( )

n = 4

0, 1, 1, 2, 3, 5, 8



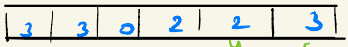
possible ways to 3 ⇒ 1+1+1
⇒ 1+2
⇒ 2+1

possible ways to 2 ⇒ 1+1
⇒ 2

ways(4) = ways(3) + ways(2)
↓ ↓
1 2

f(n) = f(n-1) + f(n-2) → fibonacci

possible ways to 4 ⇒ 1+1+1+1
1+2+1
2+1+1
1+1+2
2+2



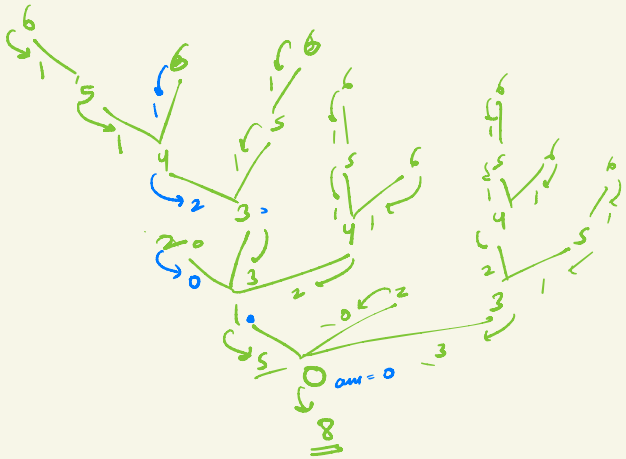
6

n^n n = arr(i)

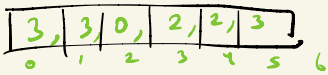
memoi 2

idx = 5
jump = 1 2

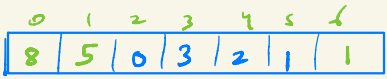
5+2




```java
public int countWays(int idx, int[] arr){
    if(idx == arr.length){
        return 1;
    }

    int ans = 0;
    for(int jump=1; jump<=arr[idx]; jump++){
        if(idx + jump <= arr.length)
            ans += countWays(idx+jump, arr);
    }

    return ans;
}
```

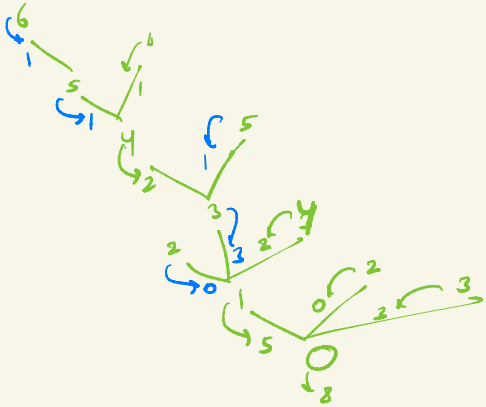


2 3
6 1
5 0
4

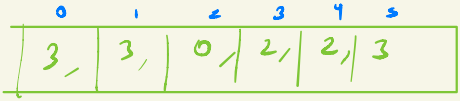


memo[0]

n



```java
public static int countWays_memo(int idx, int[] arr, int[] memo){
    if(idx == arr.length){
        return memo[idx]=1;
    }

    if(memo[idx]!=0) return memo[idx];

    int ans = 0;
    for(int jump=1; jump<=arr[idx]; jump++){
        if(idx + jump <= arr.length)
            ans += countWays_memo(idx+jump, arr,memo);
    }

    return memo[idx] = ans;
}
```

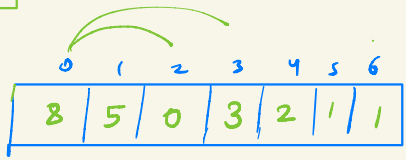


memo(4)

6

5

break till 10:45



dp(s)

memo(s)

↓

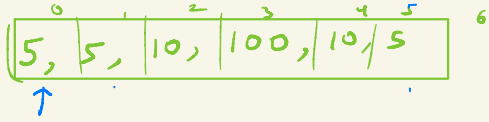
number of ways to reach n from s

idx = 6 5 4 3 2 1 0

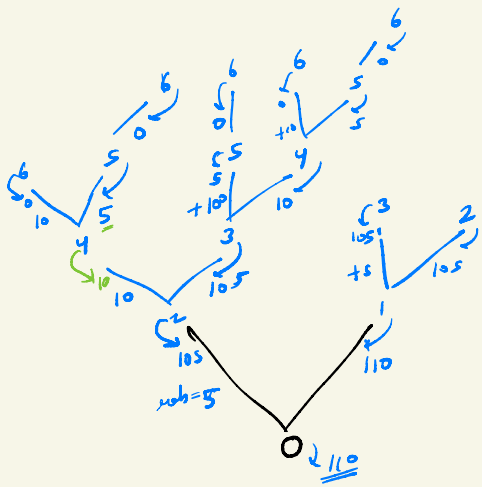
jump = 1 2 3

ans = 5 + 0 + 3 = 7

```java
public int countWays_tabulation(int idx, int[] arr, int[] memo){
    for(idx=arr.length; idx>=0; idx--){
        if(idx == arr.length){
            memo[idx]=1;
            continue;
        }

        int ans = 0;
        for(int jump=1; jump<=arr[idx]; jump++){
            if(idx + jump <= arr.length)
                ans += memo[idx+jump]; //countWays_memo(idx+jump, arr,memo);
        }

        memo[idx] = ans;

    }

    return memo[0];
}
```



105



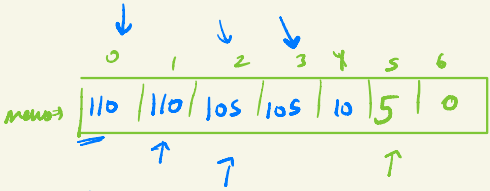

```java
public static int findMax(int idx, int[] arr){
    if(idx == arr.length){
        return 0;
    }

    int rob=arr[idx];
    int notRob=0;

    if(idx+2<=arr.length)
        rob += findMax(idx+2, arr);

    notRob = findMax(idx+1, arr);

    int ans = Math.max(rob,notRob);

    return ans;
}
```

arr → 5, 5, 10, 100, 10, 5

dry run → 9

why

rec → mem ( ) → tab ( )



dp(i) → maximum chori if we start from ith idx

r = 5 + 105

NR → 110

```java
public int findMax_tabulation(int idx, int[] arr, int[] memo){
    for(idx=arr.length; idx>=0; idx--){
        if(idx == arr.length){
            memo[idx]=0;
            continue;
        }

        int rob=arr[idx];
        int notRob=0;

        if(idx+2<=arr.length)
            rob += memo[idx+2];//findMax_memo(idx+2, arr,memo);

        notRob = memo[idx+1]; //findMax_memo(idx+1, arr,memo);

        int ans = Math.max(rob,notRob);

        memo[idx]=ans;
    }

    return memo[0];
}
```