Dynamic Programming-1

In [ ]:

```
def fun(n):
    print(n)
    fun(n-1)

fun(10)
# Infinite
#
```

In [2]:

```
def fun(n):
    if n == 0:
        return
    print(n)
    fun(n-1)

fun(10)
```

```
10
9
8
7
6
5
4
3
2
1
```

In [ ]:

```

```

In [ ]:

```
# Fibonacci series
# 0 1 1 2 3
# WAF fibb(n) -> nth fibonacci number
```

In [ ]:

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

In [ ]:

```
def fibonacciSeries(i):
    if i <= 1:
        return i
    else:
        return (fibonacciSeries(i - 1) + fibonacciSeries(i - 2))

num=10
for i in range(num):
    print(fibonacciSeries(i), end=" ")
```

In [ ]:

```
```

Fibonacci simple recursive

In [8]:

```
# TC: O(2^n)
# SC: O(n) : Recursion Stack
def fib(n):
    if n == 0 or n == 1 or n == 2:
        return n-1
    else:
        return fib(n-1) + fib(n-2)

fib(4)
```

Out[8]:

```
2
```

In [10]:

```
# TC: O(n)
# SC: O(1)
def fib(n):
    a = 0
    b = 1
    for i in range(n-1):
        c = a + b
        a = b
        b = c
    return a

fib(400)
```

Out[10]:

```
10878861746347564528976199228904974484499570547781269909975120274939392635981
6304226
```

In [ ]:

```
```

In [ ]:

```
# O(2^n)
def fib(n):
    if n == 0 or n == 1 or n == 2:
        return n-1
    else:
        return fib(n-1) + fib(n-2)

fib(4)
```

In [ ]:

Fibonacci DP: Memoization

In [22]:

```
# TC: O(n)
# SC: O(n)
# Top->Down Approach

mem = {}
# hash map :
# key-value
# int:int n:fib(n)

def fib(n):
    if n == 1 or n == 2:
        return n-1

    # memoization
    if n in mem: # O(1)
        return mem[n] # O(1)

    ans = fib(n-1) + fib(n-2)
    mem[n] = ans
    return ans

print(fib(400))
```

```
10878861746347564528976199228904974484499570547781269909975120274939392635981
6304226
```

In [21]:
```
# TC: O(n)
# SC: O(n)
# Top->Down Approach

# hash map :
# key-value
# int:int n:fib(n)
mem = {1:0, 2:1}

def fib(n):
    # memoization
    if n in mem: # O(1)
        return mem[n] # O(1)

    ans = fib(n-1) + fib(n-2)
    mem[n] = ans
    return ans

print(fib(400))
```

```
10878861746347564528976199228904974484499570547781269909975120274939392635981
6304226
```

In [ ]:
```

```

Fibonacci DP: Tabulation

In [16]:
```
table = [0 for i in range(100)] # 100 elements
print(table)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [19]:

```
# TC: O(n)
# SC: O(n)
table = [0 for i in range(100)] # 100 elements
# table = [0]*100
# table = list(range(100))

# int table[100];
def fib(n):
    table[0] = 0
    table[1] = 1

    for i in range(2,n): # 2...(n-1)
        table[i] = table[i-1] + table[i-2]

    return table[n-1]

# table 0 1 1 2 3 5 8 13
#       0 1 2 3 4 5 6 7 8 9
# i 2 3 4 5 6 7
print(fib(8))
```

```
13
```

In [ ]:

Question: 1D
[https://leetcode.com/problems/climbing-stairs/ (https://leetcode.com/problems/climbing-stairs/)](https://leetcode.com/problems/climbing-stairs/)

In [ ]:

```

class Solution {

    int[] temp = new int[45];
    public int climbStairs(int n) {
        temp[0] = 1;
        temp[1] = 2;

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

In [ ]:

```
class Solution:
    def climbStairs(self, n: int) -> int:
        if n<=2:
            return n
        table=[0]*(n+1)
        table[0]=1
        table[1]=2

        for i in range(2,n+1):
            table[i] = table[i-1]+table[i-2]
        return table[n-1]
```

In [ ]:

```
class Solution:
    def climbStairs(self, n: int) -> int:
        memo = {}
        return self.helper(n, memo)

    def helper(self, n: int, memo: dict[int, int]) -> int:
        if n == 0 or n == 1:
            return 1
        if n not in memo:
            memo[n] = self.helper(n-1, memo) + self.helper(n-2, memo)
        return memo[n]
```

In [ ]:

```
class Solution {
    unordered_map<int, int> mem = {1: 1, 2: 2};

    public:
    int climbStairs(int n) {
        unordered_map<int, int> mem;
        mem[1] = 1;
        mem[2] = 2;
        return climbStairsUtil(n, mem);
    }

    int climbStairsUtil(int n, unordered_map<int, int> &mem) {

        if (mem.count(n) != 0) {
            return mem[n];
        }

        int ans = climbStairsUtil(n-1, mem) + climbStairsUtil(n-2, mem);
        mem[n] = ans;
        return ans;
    }
};
```