Dynamic - Programming (DP)

- DP is an algorithm design technique
- "Programming" = use of tables

Toy Example

What is the sum of

$$20 + 21 + \cdots + 30 = ?$$

Toy Example

What is the sum of

$$20 + 21 + \cdots + 30 + 8 + 2$$

- DP Solves Problems by Combining Solns to subProblems.
- Storing Soln to a subproblem the first time it is solved.
- Looking up the solution when subProblem is encountered again.

DP

- The Programming refers to the use of tables (arrays) to construct a solution.
- In dynamic programming we usually reduce time by increasing the amount of space.
- We solve the problem by solving sub-problems of increasing size and saving each optimal solution in a table (usually).
- The table is then used for finding the optimal solution to larger problems.
- Time is saved since each sub-problem is solved only once.

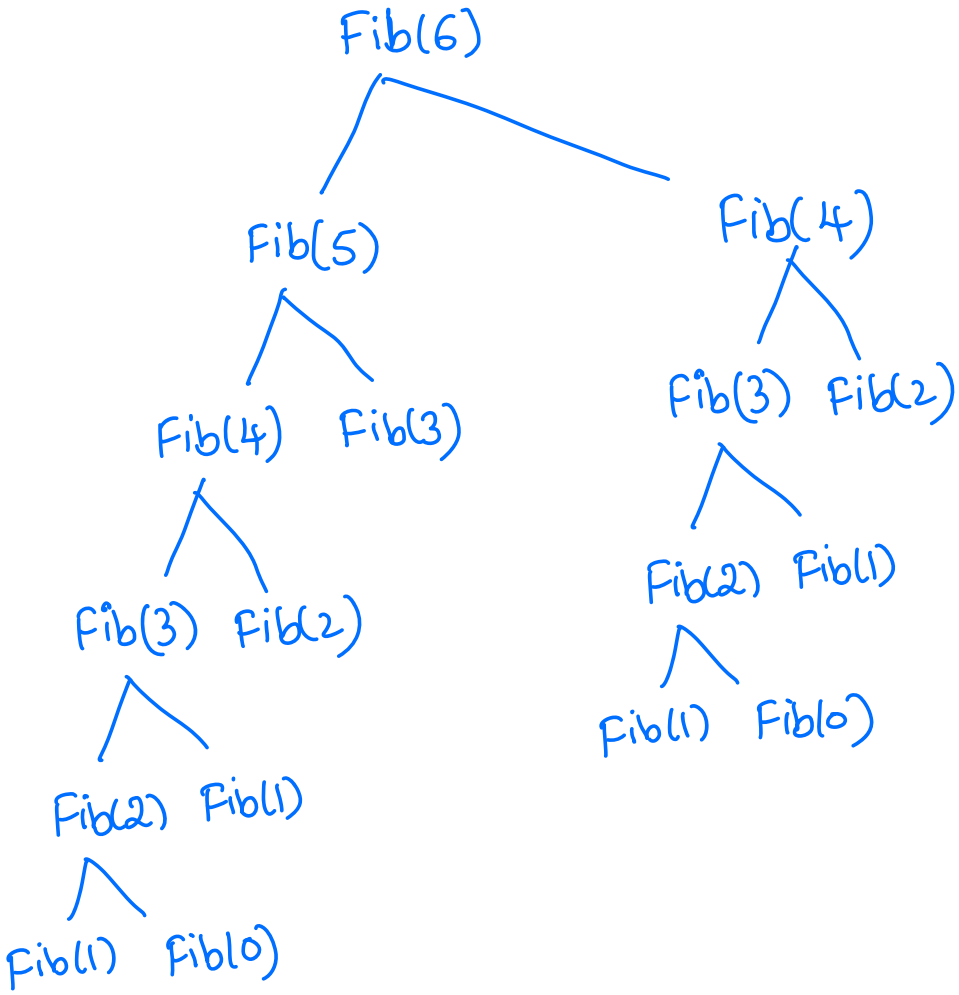
Fibonacci Numbers

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

```
Fib(n)

    If n<=1

        return 1

    else

        return Fib(n-1) + Fib(n-2)
```

Running time: $T(n) = T(n-1) + T(n-2)$

$T(n) = O(2^n)$ [Exponential]


Fib(6)
Fib(5)
Fib(4)
Fib(4)
Fib(3)
Fib(3)
Fib(2)
Fib(3)
Fib(2)
Fib(2)
Fib(1)
Fib(2)
Fib(1)
Fib(1)
Fib(0)
Fib(1)
Fib(0)

There is a Problem with the above approach?

Algorithm - I

```
def fib(n):
1    memo = {}
2    for k in range(1, n+1):
3        if k <= 2: f =1
4        else: f = memo[k-1] + memo[k-2]
5        memo[k] = f
6    return memo[n]
```

Running time : O(n)

Algorithm - II

A ← Array of size n+1 with all values set to zero

I/P: n, A

```
F(n, A)
If n ≤ 1
    return n
If A[n] != 0 then
    return A[n]
else
    A[n] = F(n-1, A) + F(n-2, A)
return A[n].
```

There are usually two equivalent ways to implement a dynamic-programming approach.

- top-down (memoization)
- bottom-up (Tabulation)

We say that the recursive procedure has been memoized;
it “remembers” what results it has computed previously.

Top-down vs Bottom-up