Dynamic Programming: Introduction

(Chapter 8)

Welcome, dear old friends: The Fibonacci numbers

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

- Each number is the sum of the previous two:

  fib(0) = 1

  fib(1) = 1

  fib(n) = fib(n-1) + fib(n-2)

- How many can we compute?

DEMO

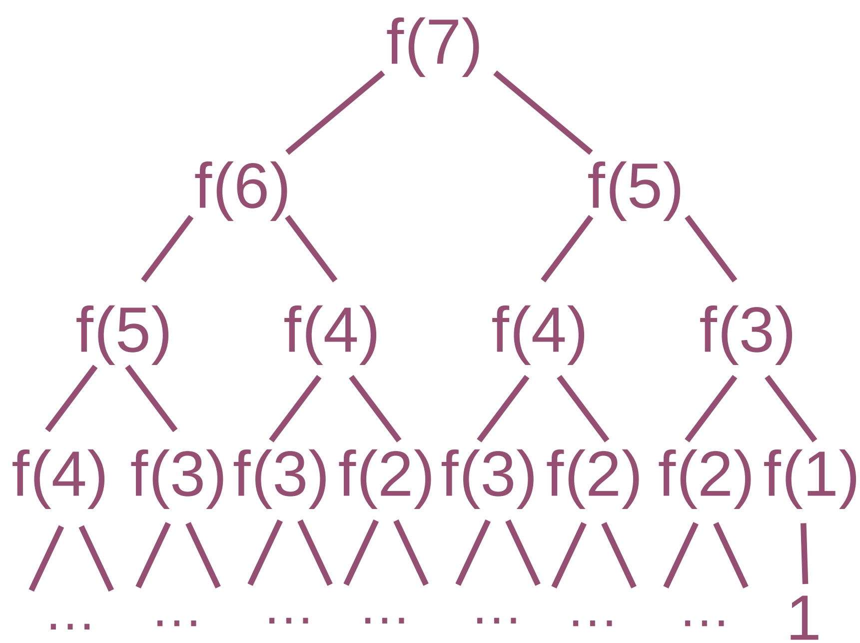
THE BASIC ALGORITHM:

```
fib (n):
    if n < 2
      return n      //i.e. fib(0) is 0 and fib(1) is 1
    else
      return = fib(n-1) + fib(n-2)
```

Fibonacci numbers: Why you so slow?

```
fib (n):
    if n < 2
      return n;
    else
       return = fib(n-1) + fib(n-2)
```

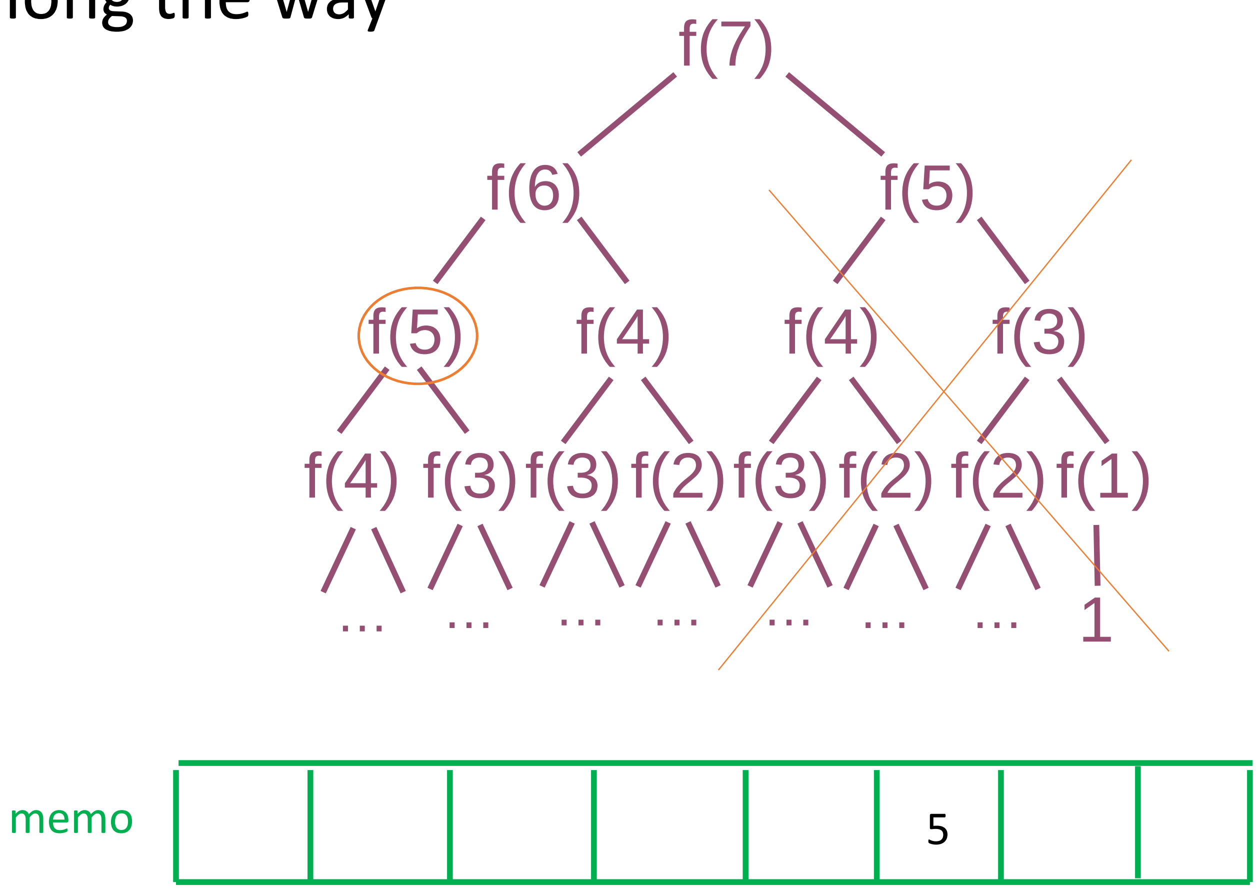
Execution tree:



F(n) takes exponential time to compute.

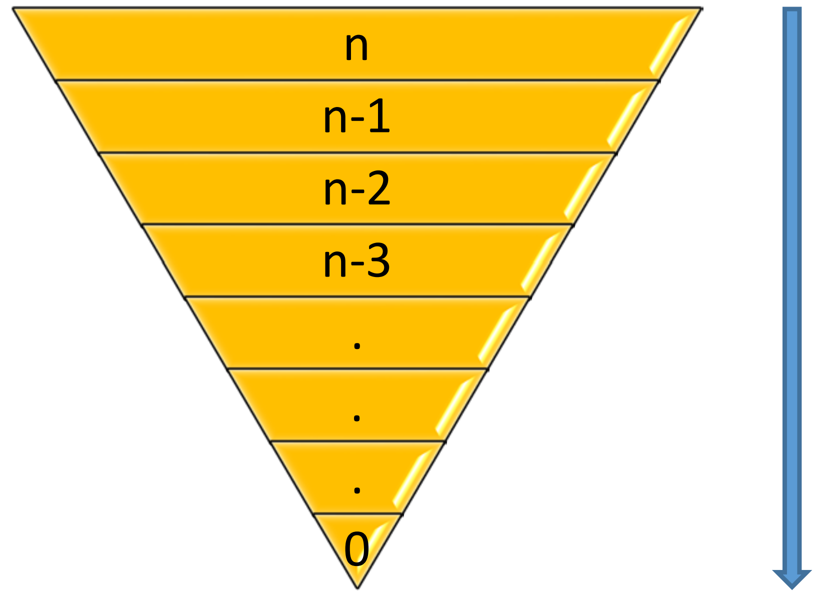
Space-time trade-off

- Augment the algorithm by *remembering* the results you get along the way



Fibs, top-down

```
fib (n) {
    if memo[n] exists, return it
    if n < 2
      return n
    else
      f = fib(n-1) + fib(n-2)
      memo[n] = f
      return f
}
```



top-down (Recursive)

memo

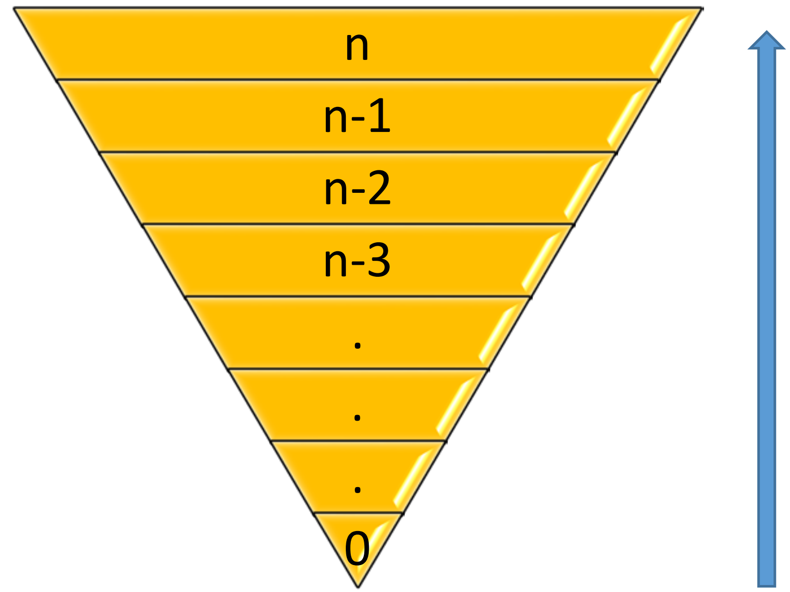
0	**1**	**1**	**. . .**	$\boldsymbol{fib(n\text{-}2)}$	$\boldsymbol{fib(n\text{-}1)}$	$\boldsymbol{fib(n)}$

Efficiency:

- time: O(n)
- space: Needs an array size O(n)

Fibs, bottom-up

```
fib (n) {
    memo[0] = 0;
    memo[1] = 1;
    for i ← 2 to n do
        memo [i] = memo[i-1] + memo[i-2]
    return memo[n]
}
```



bottom-up

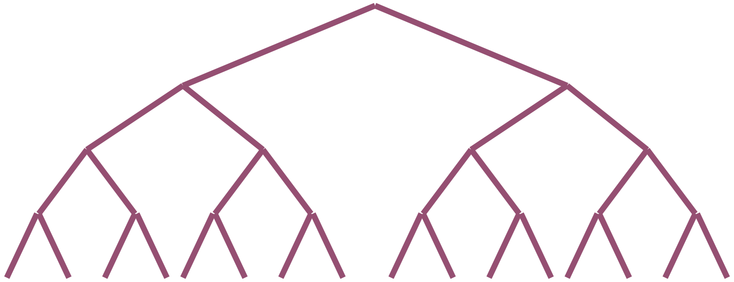
memo							
	0	**1**	**1**	. . .	***fib*(*n*-2)**	***fib*(*n*-1)**	***fib*(*n*)**

Efficiency:

- time: O(n)
- space: Needs an array size O(n)

Dynamic programming

- Key point: *remembering* recursively-defined solutions to sub-problems and using them to solve the problem
- Very much like divide-and-conquer …
  but store the solutions to sub-problems for possible reuse.
- A good idea if many of the sub-problems are repeats



Dynamic programming overview

- **Step 1:**
  - Decompose problem into simpler sub-problems
- **Step 2:**
  - Express solution in terms of sub-problems
- **Step 3:**
  - Use table to compute optimal value bottom-up
- **Step 4:**
  - Find optimal solution based on steps 1-3

Dynamic programming examples

- Fibonacci numbers
- Robot Coin Collecting
- Transitive Closure (Warshall)
- All Pairs Shortest Paths (Floyd)